UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TRABAJO FIN DE GRADO

Sistema de gestión de citas y colas para dispositivos móviles

"Diseño para la gestión y control automatizado de tiempos de espera"

Autor: Adolfo Encinas Martín

Tutores: Guillermo Suárez de Tangil, Jorge Blasco Alís

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1: Introducción

1.1. Objetivo del proyecto

El objetivo de este proyecto es desarrollar un sistema capaz de proporcionar una solución completa e integral para el control de colas de citas en servicios que requieren de una reserva previa como pueden ser la seguridad social, un punto de información o la declaración de la renta, donde es frecuente que sus usuarios soporten largas esperas sin necesidad alguna.

Así pues, este proyecto propone un sistema automatizado de avisos a dispositivos móviles de usuarios con cita reservada, mediante una comunicación unilateral entre el sistema y el usuario, en forma de notificación que indique la hora estimada de comienzo de cada cita de modo que los usuarios no tengan que esperar hasta que su cita tenga lugar.

La ventajas que presenta la utilización de un sistema de tales características frente a un posible desuso son:

- Reducción en los tiempos de espera para los usuarios.
- Agilización de los servicios.
- Agregación de un valor añadido a los servicios.

Destacar que no solamente beneficia al usuario final por no tener que esperar más tiempo del necesario para ser atendido, sino que además, cualquier posible cliente[1] que lo implante, puede ver reducidas las aglomeraciones de usuarios en el caso de un retraso en una cola de citas, u operadores parados por un adelanto sin que haya un cliente esperando en la cola.

Además, un sistema así impulsa un valor añadido diferenciador que anima a más usuarios a hacer uso del mismo.

1.2. Alcance del proyecto

La columna vertebral del proyecto será el envío automatizado de notificaciones sobre el estado del servicio. Sin embargo, esto no será posible si no se desarrolla conjuntamente una serie de herramientas de gestión para los clientes y de acción para los usuarios. A continuación, se muestra una lista de las actuaciones previstas dentro del proyecto que darán solución a ello:

[1] Compañía, administración pública, autónomo o individuo que oferta una o varias actividades en las que son necesarias una reserva previa de cita.

- ✓ **Diseño e implementación de una base de datos;** se creará una lo suficientemente ágil y potente como para almacenar y consultar grandes cantidades de información perteneciente a usuarios, trabajadores, citas, etc…

- ✓ **Diseño e implementación del back-end;** se crearán dos aplicaciones web de gestión para el cliente enfocadas a los roles: administrador y empleado, tal que el primero esté capacitado para gestionar de forma completa los servicios ofertados por el cliente y la información almacenada en la base de datos, y el segundo pueda encargarse de aquellas tareas más próximas a los usuarios como la ejecución de las citas, su reserva o cancelación. Ambas estarán alojadas en un servidor web junto a la base de datos.

- ✓ **Diseño e implementación del front-end;** el usuario dispondrá de una aplicación móvil de servicio en su smartphone donde pueda recibir las notificaciones del estado de sus citas y además tenga la capacidad de realizar diferentes acciones como reservar nuevas citas o comprobar la ruta a pie o en coche desde su posición hasta el lugar de una cita.

- ✓ **Construcción de una API;** la comunicación entre la aplicación móvil y la base de datos será universal. Esto se cumplirá gracias a la implementación de una API en el servidor, de modo que ésta reciba las peticiones realizadas desde la aplicación por un usuario, realice las operaciones necesarias de lectura o escritura en la base de datos y devuelva una respuesta en un formato multiplataforma.

- ✓ **Automatizado y control del aviso de estado del servicio;** la comprobación del estado de cada cola de citas y su notificación a los usuarios supondrá un tarea programada y periódica en el servidor.

En resumen, el proyecto supondrá el desarrollo de una base de datos, dos aplicaciones de gestión web (para administradores y empleados respectivamente), una aplicación móvil para iOS, una API y una tarea automática de notificaciones. La unión de estos elementos servirá para que el sistema automatizado de notificaciones sea lo más preciso y funcional posible.

1.3. Motivación del proyecto

La realización de este proyecto está motivada por la ganas del autor y sus tutores en intentar dar solución, en la medida de lo posible, al problema de las largas esperas en servicios con reserva de citas, aplicando los conocimientos adquiridos durante toda la formación académica.

Es necesario mencionar que lo proyectado aquí no tendría sentido sin la existencia de los smartphones. Estos dispositivos, y en especial el iPhone y el iPad (ver Figura 1), hacen que la implementación de un sistema de tales características sea más sencillo de lo que a primera vista puede pensarse.

Figura 1. Dispositivos iPhone y iPad de Apple con sistema operativo iOS.

Gracias a una tecnología sencilla y accesible para cualquier desarrollador, los smartphones poseen un canal de comunicación con el usuario a través de un sistema de notificaciones[2], que permite enviar un mensaje o alerta a cualquier dispositivo sin importar en que lugar del mundo se encuentre, convirtiéndose así en un motivo poderoso para desarrollar una plataforma como la que se describe en esta memoria.

1.4. Contenido de la memoria

La memoria contiene los siguientes capítulos:

- **Capítulo 1;** describe el objetivo, alcance y motivación del proyecto.

- **Capítulo 2;** se analizan los casos de uso del sistema, especificando los actores a intervenir, y los requisitos que debe cumplir. A continuación se realiza un análisis de las tecnologías que se han empleado durante la etapa de implementación, para finalizar con una visualización de servicios reales regidos por un sistema de reserva previa de citas que sirvan como ejemplo para la etapa de diseño e implementación.

- **Capítulo 3;** en éste se detalla el diseño de la arquitectura de los diversos componentes que componen el sistema, explicando su comportamiento a bajo nivel.

- **Capítulo 4;** queda detallada toda la implementación a nivel de código referente al módulo de control encargado de calcular el estado de las colas de citas y su posterior notificado a los usuarios.

[2] Dispositivos iOS: Apple Push Notifications Service (APNs)

- **Capítulo 5;** contiene la planificación llevada a cabo durante la realización del proyecto, el presupuesto y el modelo de negocio.

- **Capítulo 6;** presenta un plan de pruebas con los resultados obtenidos gracias a un simulador creado para la comprobación del módulo de control de colas.

- **Capítulo 7;** se detallan las conclusiones del proyecto y posibles líneas de futuro a desarrollar una vez concluido éste y la memoria.

- **Anexo;** contiene la principales referencias bibliográficas utilizadas en este documento.

CAPÍTULO 2: Análisis

2.1. Diagrama de casos de uso

Los diagramas de casos de uso son parte esencial del Lenguaje Unificado de Modelo (UML[3]). Su finalidad es mostrar de una forma gráfica las acciones que un usuario puede ejecutar dentro del sistema.

Su estructura queda definida en tres elementos básicos: actores, casos de uso y relaciones. El primero de ellos, hace referencia a los diferentes roles de usuario que interactúan con el sistema. El segundo se corresponde con cada una de las acciones individuales que permite el sistema a un actor. Mientras que el último elemento de todos indica, como bien dice su nombre, las relaciones que existen entre los actores y los casos de usos, pudiendo ser de tipo: generalización, inclusión o extensión. Para más detalles ver [2] y [3].

En los siguientes apartados se detallarán en profundidad estos tres elementos dentro del contexto de este proyecto.

2.1.1. Actores

En el ámbito de este proyecto, los actores representan a aquellos individuos que interactúan en las aplicaciones de gestión y en la de servicio. Así pues, se ha contemplado la existencia de los siguientes actores:

- **Usuario final;** actor que hará uso de la aplicación móvil de servicio desde un terminal propio, pudiendo realizar operaciones tales como reserva de citas, cancelaciones, o modificación de datos personales.

- **Empleado;** actor que ejecuta las citas reservadas por un usuario final para aquellas actividades que conformen el servicio del cliente, y que además mantiene un contrato laboral con este último. Sus tareas más importantes serán las de atender peticiones del usuario final (reserva de cita, cancelación) y comunicar al sistema el estado del servicio mediante un acción que indique cuando ha comenzado y finalizado una cita.

- **Administrador;** actor en un nivel superior al empleado, cuyas tareas más importantes serán la gestión de los cuadrantes de trabajo para los empleados, y el registro de las actividades que integran el servicio. Además, tendrá la capacidad de realizar algunas de las tareas asignadas a empleados si la situación lo requiere.

[3] Más información en [1]

2.1.2. Casos de uso

En las filas de la Tabla 1 se muestran todos los casos de uso a contemplar en el diseño del sistema, mientras que en las columnas se encuentran los actores comentados en el apartado anterior. La finalidad de dicha tabla será el listado de todo ellos y su asignación al actor correspondiente.

Casos de uso	Usuario Final	Empleado	Administrador
Activar notificación	✹		
Actualizar estado cita		✹	
Añadir actividad			✹
Añadir administrador			✹
Añadir días festivos			✹
Añadir empleado			✹
Añadir usuario final		✹	✹
Bloquear cita		✹	✹
Borrar administrador			✹
Borrar empleado			✹
Borrar usuario final		✹	✹
Cancelar cita	✹	✹	✹
Darse de alta (registro)	✹		
Darse de baja	✹		
E-mail de confirmación	✹		
Eliminar actividad			✹
Gestionar tiempo de citas			✹
Gestionar turnos de trabajo			✹
Imprimir recibo	✹		

Casos de uso	Usuario Final	Empleado	Administrador
Modificar datos personales	✹	✹	✹
Pedir cita	✹	✹	✹
Restablecer contraseña	✹	✹	✹
Ver ayuda	✹		
Ver créditos	✹		
Ver días festivos		✹	
Ver ruta	✹		
Ver tiempo de llegada	✹		
Ver turnos de trabajo		✹	
Visualizar citas	✹		
Visualizar colas	✹		

Tabla 1. Casos de uso.

La mayoría de los casos de uso pueden ser ejecutados por un único actor. No obstante, existen otros relacionados con varios actores. Por ello, se ha decidido diferenciar los casos de uso en dos tipos: específicos y comunes.

2.1.2.1. Casos de uso específicos

Se entenderá por caso de uso específico, aquél que esté relacionado solamente con un único actor en particular. Esta naturaleza implica un diseño personalizado del caso de uso, orientado en todo momento al actor que lo ejecuta.

Todos los casos de uso específicos que intervienen en este proyecto se detallan de la siguiente manera:

CU-E01: Activar notificación	
Actores	Usuario final
Descripción	El actor activa la opción de aviso con notificaciones del estado de la cola a la que pertenece la cita.
Precondiciones	El actor debe haber iniciado una sesión privada en la aplicación, y además, haber reservado una cita, o bien, estar en la etapa final de la reserva.
Postcondiciones	El sistema notifica al actor el adelanto o retraso existente en la cola de citas correspondiente.
Escenario Principal	1. Encontrarse en la etapa final de reserva de cita. 2. Activar el botón "*Activar notificaciones*".
Escenario alternativo I	1. Encontrase en la lista de citas reservadas del actor. 2. Seleccionar cita. 3. Activar el botón "*Activar notificaciones*".

Caso de uso específico 1. Activar notificación.

CU-E02: Actualizar estado cita	
Actores	Empleado
Descripción	El actor comunica al servidor que una cita ha comenzado o ha finalizado en su correspondiente turno de trabajo. Esto permite a la tarea automática de notificaciones calcular el adelanto o retraso en minutos de dicho turno y comunicarlo a los usuarios.
Precondiciones	El actor debe haber iniciado una sesión privada en la aplicación.
Postcondiciones	El sistema registra en la base de datos el comienzo o la finalización de una cita.
Escenario Principal	1. Encontrarse en la visualización de un turno de trabajo. 2. Seleccionar cita. 3. Pulsar el botón "*Comienzo*".
Escenario alternativo I	1. Encontrarse en la visualización de un turno de trabajo. 2. Seleccionar cita. 3. Pulsar el botón "*Finalizar".*

Caso de uso específico 2. Actualizar estado cita.

CU-E03: Añadir actividad	
Actores	Administrador
Descripción	El actor añade a la base de datos una actividad comprendida dentro de un servicio ofertado por el cliente, indicando sus horarios y su tiempo medio de cita. Por ejemplo: *un administrador registra la actividad "Matriculación" al servicio del punto de información de la Universidad Carlos III de Madrid.*
Precondiciones	El actor debe haber iniciado una sesión privada en la aplicación.
Postcondiciones	Una actividad es creada en la base de datos del sistema.
Escenario Principal	1. Pulsar el botón "*Gestión de actividades".* 2. Pulsar el botón "*Añadir".* 3. Introducir por teclado: un nombre de actividad y una breve descripción y la dirección del lugar. 4. Introducir los horarios de la actividad. 5. Seleccionar un tiempo medio de cita en la actividad. 6. Pulsar el botón "*Guardar".* 7. El sistema muestra un mensaje de confirmación.
Escenario alternativo I	1. El sistema informa de un error, al existir una actividad con el mismo nombre en la base de datos. 2. Repetir nuevamente el escenario principal desde el paso 3.

Caso de uso específico 3. Añadir actividad.

CU-E04: Añadir Administrador	
Actores	Administrador
Descripción	El actor añade un nuevo administrador en la base de datos
Precondiciones	El actor debe haber iniciado una sesión privada en la aplicación.
Postcondiciones	Un nuevo administrador puede hacer *login* en el sistema.
Escenario Principal	1. Pulsar el botón "*Gestión de administradores".* 2. Pulsar el botón "*Añadir".* 3. Introducir por teclado los datos del nuevo administrador. 4. Pulsar el botón "*Guardar".* 5. El sistema muestra un mensaje de confirmación.
Escenario alternativo I	1. El sistema informa de un error, al existir un registro de un administrador idéntico en la base de datos. 2. Repetir nuevamente el escenario principal desde el paso 3.

Caso de uso específico 4. Añadir administrador.

CU-E05: Añadir día festivo	
Actores	Administrador
Descripción	El actor añade un nuevo día festivo en la base de datos.
Precondiciones	El actor debe haber iniciado una sesión privada en la aplicación.
Postcondiciones	Los empleados no tendrán turnos de trabajo asignados en un día festivo. Ello implicará la ausencia de citas disponibles para su reserva en dicho día.
Escenario Principal	1. Encontrarse en el menú principal de la aplicación. 2. Pulsar el botón "*Calendario".* 3. Pulsar el botón "Festivos". 4. Pulsar el botón "*Añadir".* 5. Introducir por teclado los datos del nuevo día festivo. 6. Pulsar el botón "*Guardar".* 7. El sistema muestra un mensaje de confirmación.
Escenario alternativo I	1. El sistema informa de un error, al existir un registro de un administrador idéntico en la base de datos. 2. Repetir nuevamente el escenario principal desde el paso 4.

Caso de uso específico 5. Añadir día festivo.

CU-E06: Añadir Empleado	
Actores	Administrador
Descripción	El actor añade los datos de un empleado en la base de datos.
Precondiciones	El actor debe haber iniciado una sesión privada en la aplicación.
Postcondiciones	Se registra un empleado en la base de datos del sistema.
Escenario Principal	1. Pulsar el botón "*Gestión de empleados".* 2. Pulsar el botón "*Añadir".* 3. Introducir por teclado los datos del nuevo empleado. 4. Pulsar el botón "*Guardar".* 5. El sistema muestra un mensaje de confirmación.
Escenario alternativo I	1. El sistema informa de un error, al existir un empleado con el mismo identificador en la base de datos. 2. Repetir nuevamente el escenario principal desde el paso 3.

Caso de uso específico 6. Añadir empleado.

CU-E07: Borrar administrador	
Actores	Administrador
Descripción	El actor elimina a un administrador de la base de datos.
Precondiciones	El actor debe haber iniciado una sesión privada en la aplicación.
Postcondiciones	Se elimina del sistema el administrador seleccionado.
Escenario Principal	1. Pulsar el botón "*Gestión de administradores*". 2. Seleccionar un administrador. 3. Pulsar el botón "*Borrar*". 4. El sistema muestra un mensaje de confirmación.
Escenario alternativo I	1. El sistema informa de un error al intentar borrar el registro en la base de datos. 2. Repetir nuevamente los pasos del escenario principal.

Caso de uso específico 7. Borrar administrador.

CU-E08: Borrar empleado	
Actores	Administrador
Descripción	El actor borra los datos de un empleado de la base de datos.
Precondiciones	El actor debe haber iniciado una sesión privada en la aplicación.
Postcondiciones	Se elimina un empleado en la base de datos del sistema.
Escenario Principal	1. Pulsar el botón "*Gestión de empleados*". 2. Seleccionar un empleado. 3. Pulsar el botón "*Borrar*". 4. El sistema muestra un mensaje de confirmación.
Escenario alternativo I	1. El sistema informa de un error al intentar borrar el registro en la base de datos. 2. Repetir nuevamente los pasos del escenario principal.

Caso de uso específico 8. Borrar empleado.

CU-E09: Darse de alta (registro)	
Actores	Usuario final
Descripción	El actor registra un identificador único (correo electrónico) y una contraseña privada con los que iniciar una sesión en la aplicación móvil de servicio.
Precondiciones	N/A

CU-E09: Darse de alta (registro)	
Postcondiciones	El sistema graba en la base de datos el nuevo identificador y contraseña del actor, además de varios datos personales, con lo que ya puede iniciar la sesión.
Escenario Principal	1. Pulsar el botón "*Registrarse*". 2. Escribir un identificador único y una contraseña privada. 3. Escribir datos personales. 4. Pulsar el botón "*Hecho*". 5. El sistema muestra un mensaje de confirmación.
Escenario alternativo I	1. El sistema informa de un error, al existir un registro con igual identificador en la base de datos. 2. Repetir nuevamente el escenario principal desde el paso 2.

Caso de uso específico 9. Darse de alta (registro).

CU-E10: Darse de baja	
Actores	Usuario final
Descripción	El actor borra sus datos personales, su identificador único y su contraseña de la base de datos.
Precondiciones	El actor debe haber iniciado una sesión privada en la aplicación.
Postcondiciones	El actor ya no puede hacer iniciar sesión en la aplicación móvil de servicio.
Escenario Principal	1. Pulsar el botón "*Ajustes*". 2. Pulsar el botón "*Borrar cuenta*". 3. Pulsar el botón "*OK*" en el mensaje de alerta. 4. El sistema muestra un mensaje de confirmación.
Escenario alternativo I	1. El sistema informa de un error, al existir un registro con igual identificador en la base de datos. 2. Repetir nuevamente el escenario principal desde el paso 2.

Caso de uso específico 10. Darse de baja.

CU-E11: E-mail confirmación	
Actores	Usuario final
Descripción	El actor activa la opción de recibir un e-mail con los datos de la cita reservada en su cuenta de correo personal.
Precondiciones	El actor debe haber iniciado una sesión privada en la aplicación, y además, haber reservado una cita, o bien, estar en la etapa final de la reserva.
Postcondiciones	Un e-mail es enviado a la dirección de la cuenta de correo almacenada con la información de una cita reservada.

CU-E11: E-mail confirmación	
Escenario Principal	1. Encontrarse en la etapa final de reserva de cita. 2. Pulsar el botón "*Enviar E-mail*". 3. Confirmar envío.
Escenario alternativo I	1. Encontrase en la lista de citas reservadas del actor. 2. Seleccionar cita. 3. Pulsar el botón "*Enviar E-mail*". 4. Confirmar envío.

Caso de uso de específico 11. E-mail confirmación.

CU-E12: Eliminar actividad	
Actores	Administrador
Descripción	El actor elimina una actividad de la base de datos.
Precondiciones	El actor debe haber iniciado una sesión privada en la aplicación.
Postcondiciones	Se elimina una actividad en la base de datos del sistema.
Escenario Principal	1. Pulsar el botón "*Gestión de actividades".* 2. Seleccionar la actividad. 3. Pulsar el botón "*Borrar".* 4. El sistema muestra un mensaje de confirmación.
Escenario alternativo I	1. El sistema informa de un error al intentar eliminar el registro en la base de datos. 2. Repetir nuevamente los pasos del escenario principal desde el paso 2.

Caso de uso específico 12. Eliminar actividad.

CU-E13: Gestionar tiempo de citas	
Actores	Administrador
Descripción	El actor modifica el tiempo de cita medio en cada actividad.
Precondiciones	El actor debe haber iniciado una sesión privada en la aplicación.
Postcondiciones	El sistema ofrece al usuario final horarios de citas libres con una nueva duración media de tiempo.

CU-E13: Gestionar tiempo de citas	
Escenario Principal	1. Encontrarse en el menú principal. 2. Pulsar el botón "*Gestión de actividades".* 3. Seleccionar actividad. 4. Pulsar el botón "*Modificar*". 5. Seleccionar nuevo tiempo medio de cita. 6. Pulsar el botón "*Guardar*". 7. El sistema muestra un mensaje de confirmación.
Escenario alternativo I	1. El sistema informa de un error en la actualización de la base de datos. 2. Repetir nuevamente el escenario principal desde el paso 2.

Caso de uso específico 13. Gestionar tiempo de citas.

CU-E14: Gestionar turnos de trabajo	
Actores	Administrador
Descripción	El actor establece o modifica el horario laboral de los empleados para cada actividad en forma de turno de trabajo con duración igual a alguno de los horarios de dicha actividad. Cada turno de trabajo de un empleado supone un cola de citas a notificar para el usuario.
Precondiciones	El actor debe haber iniciado una sesión privada en la aplicación.
Postcondiciones	El sistema proporciona un horario de citas disponibles, a partir de los turnos de cada trabajador y de las citas ya reservadas en ellos.
Escenario Principal	1. Pulsar el botón *"Gestionar cuadrantes".* 2. Visualizar calendario diario de turnos por actividad. 3. Seleccionar actividad. 4. Visualizar calendario semanal de turnos para todos los empleados. 5. Marcar/desmarcar turno de trabajo para empleado. 6. Pulsar el botón *"Guardar".* 7. El sistema muestra un mensaje de confirmación.
Escenario alternativo I	1. El sistema informa de un error en la actualización de la base de datos. 2. Repetir nuevamente el escenario principal desde el paso 2.

Caso de uso específico 14. Gestionar turnos de trabajo.

CU-E15: Imprimir recibo	
Actores	Usuario final
Descripción	El actor selecciona la opción de imprimir un recibo con los datos de la cita reservada.
Precondiciones	El actor debe haber iniciado una sesión privada en la aplicación, y además, haber reservado una cita, o bien, estar en la etapa final de la reserva.

CU-E15: Imprimir recibo	
Postcondiciones	La aplicación imprime un justificante con los datos de la cita reservada.
Escenario Principal	1. Encontrarse en la etapa final de reserva de cita. 2. Pulsar el botón *"Imprimir recibo"*. 3. Seleccionar impresora. 4. Confirmar impresión. 5. El sistema muestra un mensaje de confirmación.
Escenario alternativo I	1. Encontrase en la lista de citas reservadas del actor. 2. Seleccionar cita. 3. Pulsar el botón "*Imprimir recibo*". 4. Seleccionar impresora. 5. Confirmar impresión. 6. El sistema muestra un mensaje de confirmación.
Escenario alternativo II	1. El sistema informa de un error en la conexión a la impresa. 2. Pulsar el botón *"Reintentar"*.

Caso de uso específico 15. Imprimir recibo.

CU-E16: Ver ayuda	
Actores	Usuario final
Descripción	El actor lee las directrices necesarias para entender correctamente el funcionamiento de la aplicación.
Precondiciones	El actor desconoce los pasos a seguir para gestionar sus citas una vez ha iniciado una sesión privada en la aplicación.
Postcondiciones	El actor visualiza la pantalla de ayuda.
Escenario Principal	1. Encontrarse en el menú principal de la aplicación. 2. Pulsar el botón "*Ayuda*". 3. Navegar entre las diferentes pantallas explicativas.
Escenario alternativo I	N/A

Caso de uso específico 16. Ver ayuda.

CU-E17: Ver créditos	
Actores	Usuario final
Descripción	El actor lee quién ha desarrollado la aplicación.
Precondiciones	El actor debe haber iniciado una sesión privada en la aplicación.

CU-E17: Ver créditos	
Postcondiciones	El sistema muestra en pantalla los componentes del equipo que ha desarrollado la aplicación.
Escenario Principal	1. Encontrarse en el menú principal de la aplicación. 2. Pulsar el botón "*Ajustes*". 3. Pulsar el botón "*Créditos*".
Escenario alternativo I	N/A

Caso de uso específico 17. Ver créditos.

CU-E18: Ver días festivos	
Actores	Empleado
Descripción	El actor visualiza los días de descanso general.
Precondiciones	El actor debe haber iniciado una sesión privada en la aplicación.
Postcondiciones	El sistema muestra en pantalla los días de vacaciones de los empleados en forma de calendario mensual.
Escenario Principal	1. Encontrarse en el menú principal de la aplicación. 2. Pulsar el botón "*Calendario*". 3. Pulsar el botón "Festivos". 4. Visualizar calendario mensual de vacaciones.
Escenario alternativo I	N/A

Caso de uso específico 18. Ver días festivos.

CU-E19: Ver ruta	
Actores	Usuario final
Descripción	El actor visualiza un mapa con la ruta que debe seguir para llegar correctamente al lugar de la cita.
Precondiciones	El actor debe haber iniciado una sesión privada en la aplicación.
Postcondiciones	El sistema muestra en una vista de mapa satélite la ruta de llegada a la cita.
Escenario Principal	1. Encontrarse en la lista de citas reservadas. 2. Seleccionar una cita. 3. Pulsar el botón "*Ruta*". 4. Escoger ruta a pie o en coche.

CU-E19: Ver ruta	
Escenario alternativo I	N/A

Caso de uso específico 19. Ver ruta.

CU-E20: Ver tiempo de llegada	
Actores	Usuario final
Descripción	El actor comprueba el tiempo que le acarrea llegar hasta el lugar de la cita.
Precondiciones	El actor debe haber iniciado una sesión privada en la aplicación y encontrarse en visualización de ruta en mapa.
Postcondiciones	El sistema muestra en una vista de mapa satélite la ruta de llegada a la cita con el tiempo estimado de llegada.
Escenario Principal	1. Escoger ruta a pie o en coche. 2. Leer tiempo de llegada estimado.
Escenario alternativo I	N/A

Caso de uso específico 20. Ver tiempo de llegada.

CU-E21: Ver turnos de trabajo	
Actores	Empleado
Descripción	El actor visualiza todos sus turnos de trabajo y las citas reservadas en ellos para su ejecución.
Precondiciones	El actor debe hacer iniciado una sesión privada en la aplicación.
Postcondiciones	El sistema muestra los turnos de trabajo y las citas correspondientes al actor.
Escenario Principal	1. Encontrarse en el menú principal de la aplicación. 2. Pulsar el botón "*Ver turnos de trabajo*". 3. Visualizar listado de turnos y citas.
Escenario alternativo I	N/A

Caso de uso específico 21. Ver turnos de trabajo.

CU-E22: Visualizar citas	
Actores	Usuario final
Descripción	El actor visualiza todas sus citas en pantalla.
Precondiciones	El actor debe hacer iniciado una sesión privada en la aplicación.
Postcondiciones	El sistema muestra las citas correspondientes al actor.
Escenario Principal	1. Encontrarse en el menú principal de la aplicación. 2. Pulsar el botón "*Ver citas".* 3. Visualizar listado de citas.
Escenario alternativo I	N/A

Caso de uso específico 22. Visualizar citas.

CU-E23: Visualizar colas	
Actores	Usuario final
Descripción	El actor visualiza una cola de citas con su posible adelanto o retraso.
Precondiciones	El actor debe hacer iniciado una sesión privada en la aplicación.
Postcondiciones	El sistema muestra la cola de citas de un turno de empleado en la que el actor tiene una cita reservada.
Escenario Principal	1. Encontrarse en el menú principal de la aplicación. 2. Pulsar el botón "Ver citas". 3. Seleccionar una cita. 4. Pulsar el botón "*Ver cola de citas"*.
Escenario alternativo I	N/A

Caso de uso específico 23. Visualizar colas.

2.1.2.2. Casos de uso comunes

Esta topología hace referencia a aquellos casos que están relacionados con varios actores a la vez, y que a la hora de su implementación se deberá prestar atención especial a que pueden ser ejecutados por diferentes actores.

La lista detalla de casos de uso comunes es la siguiente:

CU-C01: Añadir usuario final	
Actores	Empleado y administrador
Descripción	El actor registra un nuevo usuario final en el sistema.
Precondiciones	El actor debe haber iniciado una sesión privada en la aplicación.
Postcondiciones	Los datos de un nuevo usuario final son añadidos a la base de datos de modo que éste puede iniciar una sesión privada.
Escenario Principal	1. Pulsar el botón "*Gestión de usuarios".* 2. Pulsar el botón "*Añadir".* 3. Introducir por teclado los datos personales del usuario final. 4. Introducir el identificador de sesión del usuario final (correo electrónico). 5. Pulsar el botón "*Guardar".* 6. El sistema muestra un mensaje de confirmación.
Escenario alternativo I	1. El sistema informa de un error, al existir un usuario final con el mismo identificador en la base de datos. 2. Repetir nuevamente el escenario principal desde el paso 3.

Caso de uso común 1. Añadir usuario final.

CU-C02: Bloquear cita	
Actores	Empleado y administrador
Descripción	El actor bloquea una cita registrada en el sistema.
Precondiciones	El actor debe haber iniciado una sesión privada en la aplicación.
Postcondiciones	La cita ya entra en estado de bloqueo por lo que no podrá ser reservada nunca más.
Escenario Principal	1. Pulsar el botón "*Gestión de citas".* 2. Seleccionar la cita a bloquear. 3. Pulsar el botón "*Bloquear".* 4. Pulsar el botón "*Ok"* del mensaje de alerta. 5. El sistema muestra un mensaje de confirmación.
Escenario alternativo I	1. El sistema informa de un error, al existir un usuario final con el mismo identificador en la base de datos. 2. Repetir nuevamente el escenario principal desde el paso 3.

Caso de uso común 2. Bloquear cita.

CU-C03: Borrar usuario final	
Actores	Empleado y administrador
Descripción	El actor borra a un usuario final del sistema.
Precondiciones	El actor debe haber iniciado una sesión privada en la aplicación.
Postcondiciones	Los datos de un usuario final son borrados de la base de datos por lo que éste ya no puede iniciar una sesión privada.
Escenario Principal	1. Pulsar el botón "*Gestión de usuarios".* 2. Seleccionar un usuario final. 3. Pulsar el botón "*Borrar".* 4. Pulsar el botón "*OK"* en el mensaje de alerta. 5. El sistema muestra un mensaje de confirmación.
Escenario alternativo I	1. El sistema informa de un error. 2. Repetir nuevamente el escenario principal desde el paso 2.

Caso de uso común. Borrar usuario final.

CU-C04: Buscar usuario final	
Actores	Empleado y administrador
Descripción	El actor busca a un usuario final de entre todos los registros del sistema.
Precondiciones	El actor debe haber iniciado una sesión privada en la aplicación.
Postcondiciones	Los registros que concuerdan con los parámetros de la búsqueda son mostrados en pantalla al actor.
Escenario Principal	1. Pulsar el botón "*Gestión de usuarios".* 2. Introducir los parámetros de la búsqueda. 3. Pulsar el botón "*Buscar".* 4. Seleccionar usuario final en la lista de resultados.
Escenario alternativo I	N/A

Caso de uso común 4. Buscar usuario final.

CU-C05: Cancelar cita	
Actores	Usuario final, empleado y administrador
Descripción	El actor cancela la reserva de una cita.
Precondiciones	El actor debe haber iniciado una sesión privada en la aplicación.

CU-C05: Cancelar cita	
Postcondiciones	La reserva de una cita es cancelada pasando a estar nuevamente disponible.
Escenario Principal	1. Encontrarse en el menú principal de la aplicación. 2. Pulsar el botón "*Ver citas*" (usuario final) o "*Gestión de citas*" (empleado y administrador). 3. Seleccionar cita. 4. Pulsar el botón "*Cancelar*". 5. Pulsar el botón "*OK*" en el mensaje de alerta. 6. El sistema muestra un mensaje de confirmación.
Escenario alternativo I	1. El sistema informa de un error al intentar cancelar la reserva. 2. Repetir nuevamente los pasos del escenario principal.

Caso de uso común 5. Cancelar cita.

CU-C06: Modificar datos personales	
Actores	Usuario final, empleado y administrador
Descripción	El actor modifica los datos personales de un usuario final. En el caso del empleado o administrador, es necesaria una autorización previa del usuario final.
Precondiciones	El actor debe haber iniciado una sesión privada en la aplicación.
Postcondiciones	El sistema guarda los cambios en la base de datos.
Escenario Principal	1. Encontrarse en el menú *"Ajustes"* (Usuario final) o en la vista de datos personales de un usuario final (empleado y administrador). 2. Pulsar el botón *"Editar"*. 3. Introducir nuevos datos en los campos disponibles. 4. Pulsar el botón *"Guardar"*. 5. El sistema muestra un mensaje de confirmación.
Escenario alternativo I	1. El sistema informa de un error al actualizar la información en la base de datos. 2. Repetir nuevamente el escenario principal desde el paso 3.

Caso de uso común 6. Modificar datos personales.

CU-C07: Pedir cita	
Actores	Usuario final, empleado y administrador
Descripción	El actor reserva una cita para él mismo en el caso caso de un usuario final, y para este último en el caso de que el actor sea un empleado o administrador. La reserva de una cita conlleva dos pasos: reservar y confirmar. En el primero la cita estará reservada para el usuario final durante un período máximo de 5 minutos. Cuando ésta sea confirmada la cita estará reservada para el usuario final indefinidamente.
Precondiciones	El actor debe haber iniciado una sesión privada en la aplicación. En el caso de empleado y administrador haber seleccionado previamente un usuario final para reservarle una cita.
Postcondiciones	La cita reservada y confirmada está asociada al usuario final indefinidamente.
Escenario Principal	1. Pulsar el botón "*Pedir Cita*". 2. Escoger cita disponible para una actividad para una fecha con turno de trabajo o cola de citas establecido. 3. Pulsar el botón "*Reservar*". 4. El sistema muestra un mensaje de confirmación. 5. Ver datos de la cita reservada. 6. Pulsar el botón "*Confirmar*". 7. El sistema muestra un mensaje de confirmación. 8. Ver datos de la cita reservada y confirmada. 9. Habilitar opciones.
Escenario alternativo I	1. El sistema informa de un error en el proceso de reserva. 2. Repetir nuevamente el escenario principal desde el paso 2.

Caso de uso común 7. Pedir cita.

CU-C08: Restablecer contraseña	
Actores	Usuario final, empleado y administrador
Descripción	El actor obtiene una nueva contraseña, que sustituye a la original, con la que iniciar una sesión privada.
Precondiciones	El actor ha introducido su identificador en el menú de inicio de sesión pero no recuerda su contraseña.
Postcondiciones	El sistema almacena una nueva contraseña en la base de datos donde se encuentran toda la información del actor.

CU-C08: Restablecer contraseña	
Escenario Principal	1. Encontrarse en la pantalla de bienvenida de la aplicación (aplicación móvil de servicio cuando el actor es un usuario final y la respectiva aplicación web de gestión en el caso de un empleado o administrador). 2. Pulsar el botón "*Olvidé la contraseña"*. 3. Introducir por teclado el identificador único (correo electrónico). 4. Pulsar el botón "*Restablecer contraseña"*. 5. Confirmar el siguiente mensaje: "Se ha enviado un e-mail a su cuenta personal de correo, con un enlace para restablecer su contraseña". 6. Dirigirse a la bandeja de entrada del correo electrónico personal. 7. Abrir el e-mail enviado por el sistema. 8. Pinchar en el enlace proporcionado para el restablecimiento de la contraseña. 9. Rellenar el formulario con la nueva contraseña. 10. Pulsar el botón *"Guardar"*. 11. El sistema muestra un mensaje de confirmación.
Escenario alternativo I	1. El sistema informa de un error en caso de no guardarse correctamente la nueva contraseña. 2. Repetir nuevamente el escenario principal.

Caso de uso común 8. Restablecer contraseña.

2.1.3. Relaciones

Tras una detallada explicación de todos los casos de uso, las relaciones de generalización, extensión o inclusión de éstos con los diferentes actores no han sido definidas hasta el momento. La manera más sencilla de entenderlas con claridad, es visualizar los diagramas de casos de uso mostrados a continuación:

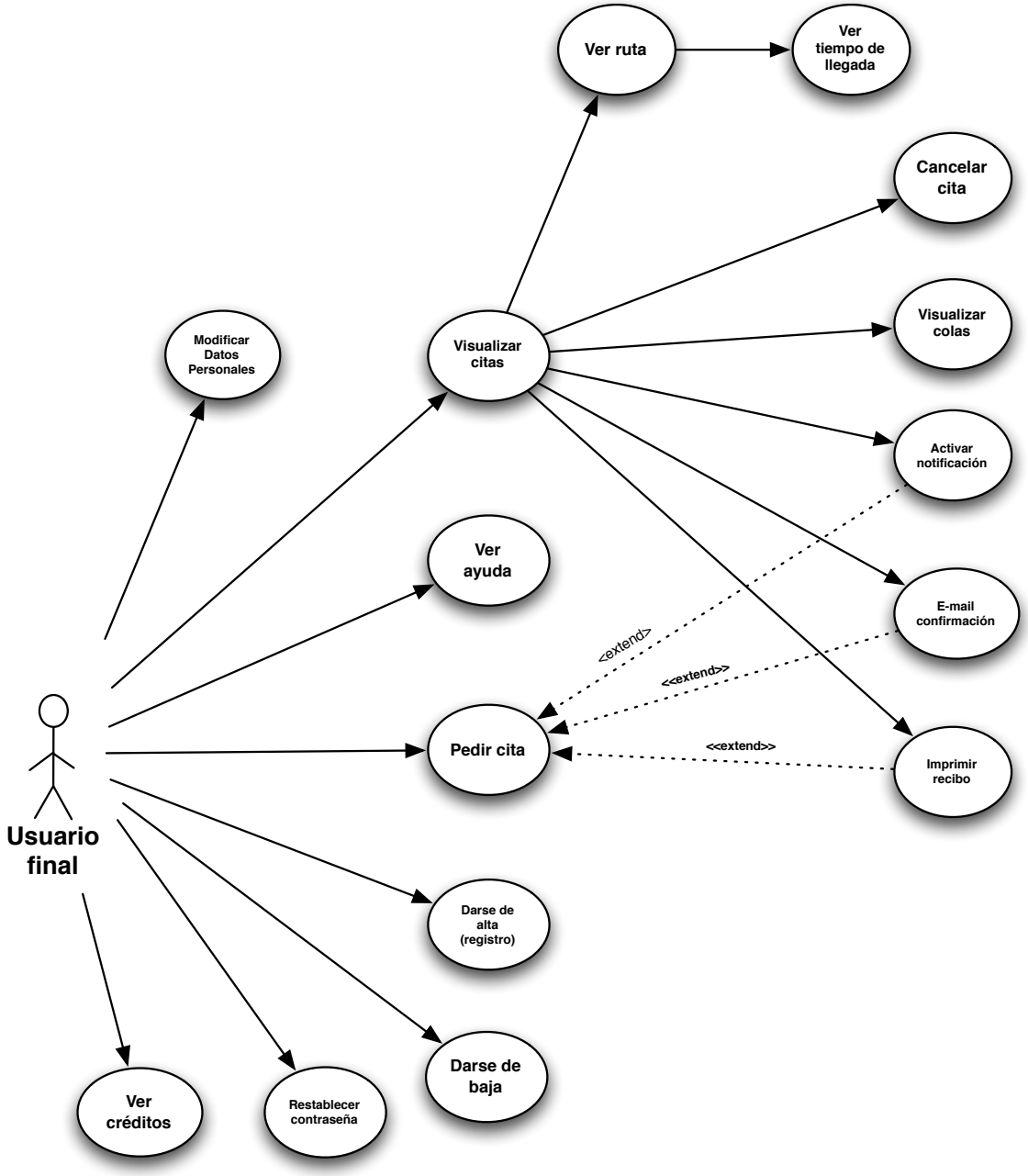

Figura 2. Casos de uso relacionados con el actor usuario final.

En la Figura 2, se relacionan los casos de uso pertenecientes al actor "*usuario final*". Existen ocho casos de uso principales (modificar datos personales, visualizar citas, ver ayuda, pedir cita, darse de alta, darse de baja, restablecer contraseña y ver créditos)

de los que en ocasiones se extienden o continúan otros casos secundarios. Al hablar de extensión se entiende que el caso de uso es una operación opcional para el actor.

En la Figura 3, se muestra la relación gráfica de los casos de uso que pueden ser ejecutados por el empleado y el administrador.

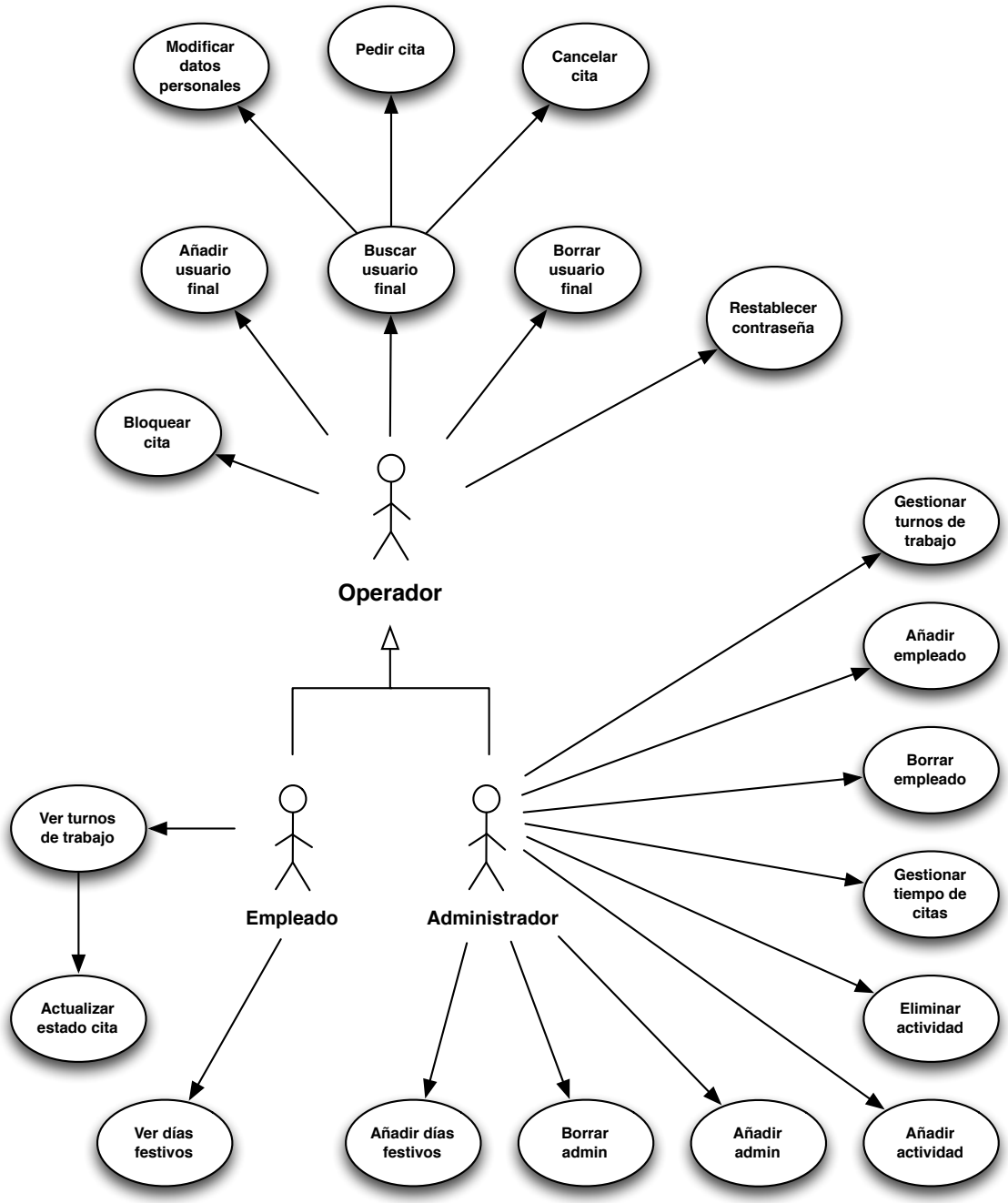

Figura 3. Casos de uso relacionados con los actores empleado y administrador.

Debido a la gran cantidad de casos de uso comunes para ambos actores, es necesario representar un actor con un carácter abstracto, denominado operador, que unifique todas esas tareas. Así pues, queda reflejado que el administrador ejecutará sus propias

tareas además de muchas otras correspondientes al empleado, siempre desde su propia aplicación web de gestión.

2.2. Pliego de requisitos

En este apartado se detalla cada uno de los requisitos funcionales y no funcionales que deberá cumplir el sistema una vez implementado y testado. Los requisitos funcionales comprenderán las distintas operaciones que realiza el software, como respuesta a la interacción del usuario con el sistema. Mientras que los no funcionales, abarcarán las diferentes tecnologías utilizadas, tanto en hardware como en software durante todo el desarrollo.

2.2.1. Requisitos funcionales

Identificador	RF-01
Descripción	La aplicación correspondiente al usuario final, empleado o administrador permitirá el inicio de una sesión privada, mediante la petición de un identificador único (correo electrónico) y una contraseña.
Prioridad	Alta
Verificabilidad	Alta

Requisito Funcional 01: Inicio de una sesión.

Identificador	RF-02
Descripción	La aplicación ofrecerá la posibilidad de crear una cuenta a un usuario final en la aplicación móvil de servicio con un identificador único y una contraseña para poder iniciar una sesión.
Prioridad	Alta
Verificabilidad	Alta

Requisito Funcional 02: Registro de un usuario final en el sistema.

Identificador	RF-03
Descripción	La activación de una cuenta de usuario final, como último paso en el registro del mismo dentro del sistema, consistirá en la confirmación de la veracidad de los datos personales mediante un enlace web enviado dentro un e-mail a la cuenta de correo electrónico introducida en el formulario de registro.
Prioridad	Alta
Verificabilidad	Alta

Requisito Funcional 03: Activación de la cuenta del usuario final.

Identificador	RF-04
Descripción	Para que el usuario final, empleado o administrador pueda restablecer su contraseña, la aplicación móvil de servicio o web de gestión correspondiente, requerirá la dirección de correo electrónico utilizada como identificador único para enviar un e-mail que contenga un enlace directo al formulario de restablecimiento de la contraseña, que contendrá dos campos de texto en los que introducir la nueva contraseña y su duplicado (método de confirmación estándar).
Prioridad	Alta
Verificabilidad	Alta

Requisito Funcional 04: Restablecimiento de la contraseña.

Identificador	RF-05
Descripción	La aplicación móvil de servicio proporcionará al usuario final una explicación detallada de su funcionamiento.
Prioridad	Media
Verificabilidad	Alta

Requisito Funcional 05: Pantalla de ayuda.

Identificador	RF-06
Descripción	La aplicación móvil de servicio permitirá reservar una cita a un usuario final dentro de la actividad que requiera y en la fecha y hora que más le convenga, dentro de las disponibles. A su vez, un empleado o un administrador, desde su aplicación web de gestión correspondiente, podrá reservar una cita para un usuario final en caso de que a este último no le sea posible realizarlo desde su dispositivo móvil.
Prioridad	Alta
Verificabilidad	Alta

Requisito Funcional 06: Pedir cita.

Identificador	RF-07
Descripción	La aplicación móvil de servicio permitirá al usuario final activar el aviso automatizado con notificaciones a su dispositivo móvil sobre el estado de la cola de cada una de sus citas de manera individual. Esto será posible en la etapa de confirmación de una reserva, y dentro de la vista de detalle de cada una de las citas reservadas en espera de ejecución.
Prioridad	Alta
Verificabilidad	Alta

Requisito Funcional 07: Activación de notificaciones.

Identificador	RF-08
Descripción	La aplicación móvil de servicio permitirá a un usuario final imprimir un recibo con los detalles de una cita. Esto será posible en la etapa de confirmación de una reserva, y dentro de la vista de detalle de cada una de las citas reservadas en espera de ejecución.
Prioridad	Media
Verificabilidad	Alta

Requisito Funcional 08: Impresión de recibo.

Identificador	RF-09
Descripción	La aplicación móvil de servicio permitirá a un usuario final recibir un E-mail con los datos de una cita (recibo) en su cuenta personal de correo. Esto será posible en la etapa de confirmación de una reserva, y dentro de la vista de detalle de cada una de las citas reservadas en espera de ejecución.
Prioridad	Media
Verificabilidad	Alta

Requisito Funcional 09: Envío de un mail de confirmación.

Identificador	RF-10
Descripción	La aplicación permitirá al cliente visualizar cada una de sus citas, indicando los días restantes hasta su ejecución, y la actividad a la que pertenecen.
Prioridad	Alta
Verificabilidad	Alta

Requisito Funcional 10: Visualización de citas.

Identificador	RF-11
Descripción	La aplicación móvil de cliente permitirá al cliente visualizar las colas de citas dónde tenga reservada una cita. Por otro lado, un empleado o administrador, desde su correspondiente aplicación web de servicio, visualizará todas las colas de citas en su forma equivalente de turnos de trabajo por empleado.
Prioridad	Alta
Verificabilidad	Alta

Requisito Funcional 11: Mostrar colas de citas o turnos de trabajo.

Identificador	RF-12
Descripción	Las aplicaciones de gestión y de servicio permitirán cancelar citas a un empleado o administrador, y a un usuario final respectivamente. En el primero de los casos, el empleado o administrador decidirá si la cita cancelada pasará a estar disponible nuevamente para su reserva, o bien, quedará bloqueada por ser imposible su ejecución.
Prioridad	Alta
Verificabilidad	Alta

Requisito Funcional 12: Cancelar citas.

Identificador	RF-13
Descripción	La aplicación móvil de servicio mostrará una ruta desde el punto en el que se encuentre el usuario final hasta la localización en la que tenga lugar una de sus citas.
Prioridad	Media
Verificabilidad	Alta

Requisito Funcional 13: Ruta de llegada hasta la cita.

Identificador	RF-14
Descripción	La aplicación móvil de servicio permitirá a un usuario final modificar cada uno de los datos personales registrados en su cuenta. De la misma manera, un empleado o administrador podrá cambiar los datos personales de un usuario final desde su correspondiente aplicación web de gestión, siempre a petición de este último.
Prioridad	Alta
Verificabilidad	Alta

Requisito Funcional 14: Modificación de perfil.

Identificador	RF-15
Descripción	La aplicación móvil de servicio mostrará al usuario final los créditos de la misma.
Prioridad	Baja
Verificabilidad	Alta

Requisito Funcional 15: Mostrar créditos.

Identificador	RF-16
Descripción	Una vez finalizado el proceso completo de reserva con confirmación de una cita en la aplicación móvil de servicio por parte de un usuario final, se establecerá un recordatorio en el dispositivo, independientemente de si la sesión de éste finaliza o no. De esta manera, el dispositivo avisará de la proximidad de una cita con una hora de antelación. Este mecanismo no guarda relación alguna con la recepción automática de notificaciones en el dispositivo sobre el estado de una cola de citas.
Prioridad	Media
Verificabilidad	Baja

Requisito Funcional 16: Establecimiento automático de los recordatorios.

Identificador	RF-17
Descripción	El sistema presentará una tarea periódica y automática que enviará mensajes mediante un sistema de notificaciones al usuario final cuando éste tenga una reserva en cualquiera de las colas de citas que estén siendo ejecutada por empleados. De esta manera, el usuario final conocerá con alertas en su dispositivo móvil (previa adquisición e instalación de la aplicación de servicio) el posible adelanto o retraso que llevan sus citas, conociendo así la hora a la que debe presentarse en el lugar de la actividad correspondiente.
Prioridad	Media
Verificabilidad	Alta

Requisito Funcional 17: Tarea automática de notificaciones sobre el estado de una cola de citas.

Identificador	RF-18
Descripción	La aplicación móvil de servicio permitirá al usuario final comprobar el tiempo estimado de llegada al lugar de una cita desde su posición actual, ya sea a pie o en coche.
Prioridad	Media
Verificabilidad	Alta

Requisito Funcional 18: Tiempo estimado de llegada a la cita.

Identificador	RF-19
Descripción	La aplicación web de gestión para empleados suministrará un mecanismo para que cada uno de ellos adviertan al sistema de que han comenzado o finalizado una cita (cambio de estado) dentro de un turno de trabajo. Ésta será la referencia para que la tarea automática de envío de notificaciones con el estado de una cola de citas (turno de trabajo) calcule el adelanto o retraso de la misma.
Prioridad	Alta
Verificabilidad	Alta

Requisito Funcional 19: Actualización del estado de citas.

Identificador	RF-20
Descripción	La aplicación web de gestión correspondiente permitirá a cada empleado y administrador buscar citas, según diferentes criterios: usuario, fecha y hora, empleado, estado o actividad.
Prioridad	Alta
Verificabilidad	Alta

Requisito Funcional 20: Búsqueda de citas.

Identificador	RF-21
Descripción	La aplicación web de gestión para administradores permitirá a éstos introducir o borrar empleados en la base de datos.
Prioridad	Alta
Verificabilidad	Alta

Requisito Funcional 21: Gestión de los empleados.

Identificador	RF-22
Descripción	La aplicación web de gestión para administradores permitirá a éstos introducir, modificar o borrar las actividades que integren un servicio del cliente.
Prioridad	Alta
Verificabilidad	Alta

Requisito Funcional 22: Gestión de las actividades.

Identificador	RF-23
Descripción	La aplicación web de gestión para administradores permitirá a éstos ajustar el tiempo medio de duración por cita en cada actividad. Dicha cantidad será una estimación y servirá como parámetro para generar equitativamente las citas disponibles en sus correspondientes colas. Su modificación tendrá efecto cuando se creen nuevos turnos de trabajos para los empleados, es decir, en nuevas colas de citas.
Prioridad	Alta
Verificabilidad	Baja

Requisito Funcional 23: Establecimiento del tiempo de cita medio por actividad.

Identificador	RF-24
Descripción	Un administrador desde su aplicación web de gestión correspondiente gestionará la creación y destrucción de turnos de trabajo por empleado y actividad, y cuya duración será idéntica a cualquiera de los horarios que componen la actividad. Un turno de trabajo supondrá una cola de citas, de modo que cuando sean creados, automáticamente se generarán las citas correspondientes con estado "*disponible*".
Prioridad	Alta
Verificabilidad	Media

Requisito Funcional 24: Gestión de los turnos de trabajo de los empleados.

Identificador	RF-25
Descripción	La existencia de citas con estado "*disponible*" dependerá de los turnos de trabajo existentes y de las citas ya reservadas.
Prioridad	Alta
Verificabilidad	Media

Requisito Funcional 25: Citas disponibles.

Identificador	RF-26
Descripción	La aplicación web de gestión para administradores permitirá a un administrador introducir o borrar administradores en la base de datos, con la condición de que su cuenta no podrá ser borrada por uno mismo.
Prioridad	Media
Verificabilidad	Alta

Requisito Funcional 26: Gestión de administradores.

2.2.2. Requisitos no funcionales

Identificador	RNF-01
Descripción	La aplicación móvil de servicio para los usuarios finales será desarrollada enteramente en lenguaje Objective-C dentro del SDK para iOS propiedad de Apple.
Prioridad	Alta
Verificabilidad	Alta

Requisito no funcional 01: Lenguaje de programación de la aplicación móvil.

Identificador	RNF-02
Descripción	El servidor utilizado en el establecimiento de la base de datos y en el funcionamiento lógico tanto de las aplicaciones web de gestión como de la API utilizada para la comunicación con la aplicación móvil de servicio, será programado con el Framework Ruby on Rails 3.1. Además se hará uso del lenguaje HTML y de las tecnologías CSS y JavaScript para el contenido visual.
Prioridad	Alta
Verificabilidad	Alta

Requisito no funcional 02: Lenguajes de programación del servidor y aplicaciones web.

Identificador	RNF-03
Descripción	La aplicación móvil de servicio podrá ser ejecutada en dispositivos iPad, iPhone o iPod Touch con iOS 4.3 o superior. Las aplicaciones web de gestión ofrecerán total compatibilidad con navegadores web Chrome, Safari o Firefox.
Prioridad	Alta
Verificabilidad	Alta

Requisito no funcional 03 :Dispositivos.

Identificador	RNF-04
Descripción	La aplicación móvil de servicio deberá seguir las directrices marcadas en el dossier "Apple Human Interface Guidelines" [5] para poder ser aceptada en el AppStore.
Prioridad	Alta
Verificabilidad	Baja

Requisito no funcional 04: Apple Human Interface Guidelines.

Identificador	RNF-05
Descripción	El idioma por defecto de todas las aplicaciones será el castellano.
Prioridad	Alta
Verificabilidad	Alta

Requisito no funcional 05: Idioma castellano.

Identificador	RNF-06
Descripción	El segundo idioma disponible será el inglés.
Prioridad	Media
Verificabilidad	Alta

Requisito no funcional 06: Idioma inglés.

Identificador	RNF-07
Descripción	Como medida de seguridad, el servidor llevará instalado un certificado SSL (Secure Socket Layer) para que todas las peticiones, consultas y comunicaciones se realicen mediante el protocolo de capa de conexión segura.
Prioridad	Alta
Verificabilidad	Alta

Requisito no funcional 07: Seguridad servidor.

Identificador	RNF-08
Descripción	La base de datos establecida en el servidor será de tipo PostgreSQL.
Prioridad	Alta
Verificabilidad	Alta

Requisito no funcional 08:Topología de la base de datos.

Identificador	RNF-09
Descripción	La seguridad de los datos personales del usuario se asegurará con el sistema de funciones Hash criptográficas SHA-256.
Prioridad	Alta
Verificabilidad	Alta

Requisito no funcional 09: Seguridad en la base de datos.

Identificador	RNF-10
Descripción	La ruta hasta el lugar de la cita hará uso de la API de Google Maps.
Prioridad	Media
Verificabilidad	Alta

Requisito no funcional 10: Tecnología para las rutas en la aplicación móvil.

Identificador	RNF-11
Descripción	El diseño del servidor será los más escalable posible permitiendo la extensión a otras tecnologías móviles y web en forma de nuevas aplicaciones de gestión y de servicio.
Prioridad	Alta
Verificabilidad	Media

Requisito no funcional 11: Escalabilidad.

Identificador	RNF-12
Descripción	En una primera instancia existirá una única cuenta de administrador cuyos datos identificadores (correo electrónico y contraseña) serán proporcionados al cliente.
Prioridad	Alta
Verificabilidad	Alta

Requisito no funcional 12: Administrador por defecto.

Identificador	RNF-13
Descripción	En caso de cancelación de una cita por parte de un empleado o administrador, el sistema enviará automáticamente al usuario final una notificación a su dispositivo móvil y un e-mail a su dirección de correo registrado.
Prioridad	Alta
Verificabilidad	Alta

Requisito no funcional 13: Envío de e-mail y modificación por cancelación de una cita.

Identificador	RNF-14
Descripción	Las notificaciones del estado de las colas de citas deberán pasar a través del APNS (Apple Push Notification Server) para una llegada correcta a los dispositivos móviles de los usuarios finales.
Prioridad	Alta
Verificabilidad	Alta

Requisito no funcional 14: Envío adecuado de las notificaciones.

Identificador	RNF-15
Descripción	Una vez que el usuario final haya iniciado una sesión privada en la aplicación móvil de servicio, ésta enviará automáticamente al servidor dos identificadores alfanuméricos: el device_token y el uuid. El primero permitirá al servidor APNS (Apple Push Notifications Server) redireccionar las notificaciones enviadas desde el servidor del sistema a los dispositivos adecuados. El segundo supondrá un identificador único invariante por dispositivo para guardar una relación de los dispositivos y sus usuarios.
Prioridad	Alta
Verificabilidad	Alta

Requisito no funcional 15: Identificadores alfanuméricos.

Identificador	RNF-16
Descripción	En caso de que una misma aplicación móvil de servicio (un único dispositivo) sea utilizada por dos o más usuarios finales diferentes, las notificaciones sobre el estado de las colas de citas se corresponderán con aquellas que vayan dirigidas al usuario final que mantenga una sesión abierta en ella, o en su defecto, al último que haya iniciado y cerrado una sesión.
Prioridad	Alta
Verificabilidad	Alta

Requisito no funcional 16: Uso de un única aplicación móvil por varios usuarios finales.

2.3. Análisis tecnológico

En el diseño del sistema, se ha considerado dividir a éste en dos capas fácilmente diferenciables para asegurar una correcta implementación del mismo: una primera capa, denominada "*de aplicación",* que concentre todo lo referente a la aplicación móvil de servicio, donde los usuarios finales puedan gestionar todas sus citas desde cualquier lugar con sólo disponer de una conexión Wi-Fi o 3G; una segunda "*capa de servidor"* en la que se agrupen las aplicaciones web de gestión para empleados y administradores, la base de datos donde almacenar información, la lógica de negocio y la API móvil necesaria para la comunicación entre ambas capas.

El mayor peso del trabajo en el sistema recaerá sobre la *capa de servidor* pues es ahí donde se ejecutará toda la lógica de negocio y donde se realizarán las correspondientes consultas y escrituras de registros en la base de datos. Además, gestionará la tarea automática de envío de notificaciones y dará respuesta a las peticiones realizadas desde la aplicación móvil con la API móvil.

Por otro lado, la *capa de aplicación* representará únicamente la interfaz necesaria para que un usuario final interactúe con el sistema. Su objetivo es mostrar los resultados de las consultas y las escrituras en la base de datos.

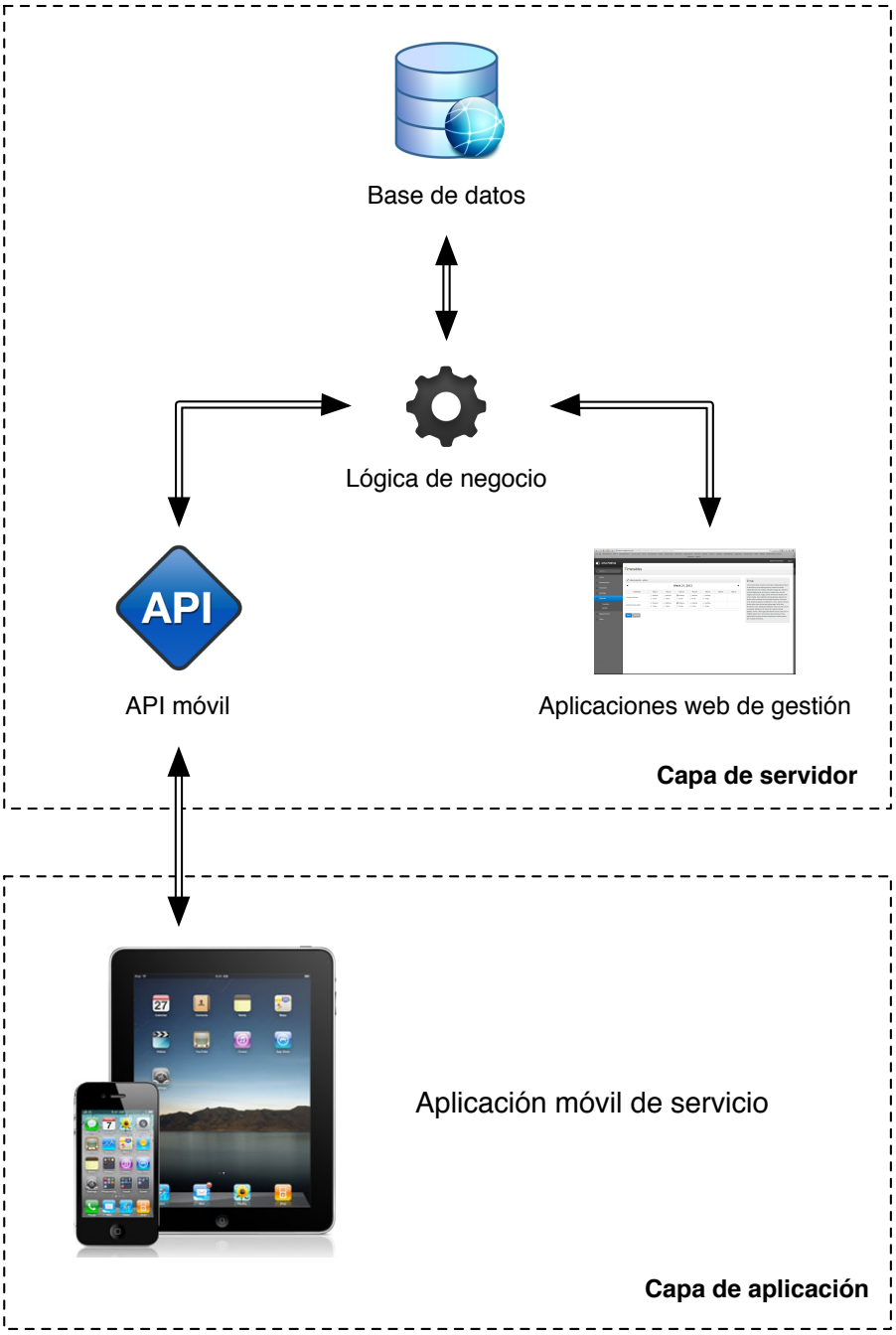

Figura 4: Capas del sistema[4].

En la Figura 4, se muestra de manera gráfica la división del sistema en dos capas, de modo que su estructura pueda ser visualizada con mayor claridad.

En los siguientes apartados, se detallarán todas las tecnologías a tener en cuenta en cada una de las capas.

[4] Imágenes obtenidas en [17]

2.3.1. Tecnologías en la capa de aplicación

La aplicación móvil de servicio creada en este proyecto debe de funcionar de la mejor manera posible en un dispositivo móvil, de modo que el usuario final pueda reservar una cita desde cualquier lugar del mundo. Es por ello que es necesario estudiar aquellas plataformas móviles (ver figura 5) que ofrezcan un buen rendimiento y facilidad en el desarrollo.

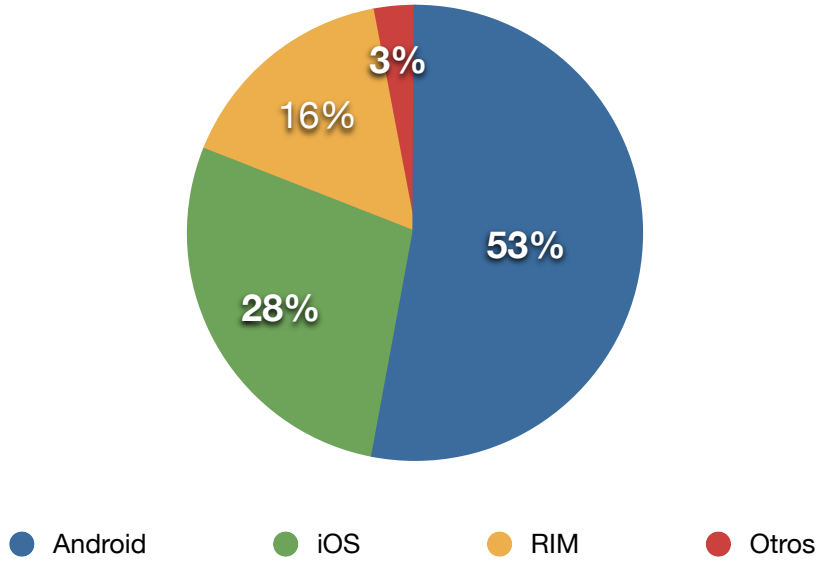

Figura 5: Cuota de mercado mundial de sistemas operativos móviles[5].

Como se puede observar en la Figura 5, actualmente las plataformas móviles más importantes son Android, iOS y RIM (Research In Motion). Esto da una primera idea de la dirección que debe tomar el estudio de cada una de ellas, pues aquellas que estén más extendidas serán las que ofrezcan un mayor número de descargas y, por lo tanto, de ingresos.

Para no extenderse demasiado en un estudio exhaustivo de todas y cada una de ellas, se ha decidido concentrarse en las dos más importantes: Android y iOS. A continuación se detallarán los diferentes aspectos que las caracterizan para finalmente concluir con la elección de una de ellas, hecho que marcará el desarrollo general del proyecto.

2.3.1.1. Software de desarrollo

Tanto en iOS como en Android, existen una gran cantidad de recursos que facilitan el desarrollo de aplicaciones destinadas a terminales móviles compatibles. Los más conocidos y usados son los SDK (Software Development Kit) que proporcionan Apple y Google de manera totalmente gratuita en [13] y [14] respectivamente.

[5] Datos obtenidos en [25].

En el caso de la plataforma iOS de Apple, el SDK integra numerosas herramientas con las que programar y depurar código. Las más utilizadas, y con las que el desarrollador tendrá que convivir continuamente, son: Xcode, iOS Simulator, e Instruments. La primera de ellas, es el IDE principal donde se escribe el código de la aplicación en lenguaje de alto nivel Objective-C, se compila y se depura. El iOS Simulator ayuda a testear la aplicación en la propia computadora de desarrollo, en caso de no disponer de un dispositivo iOS. Por último, Instruments permite conocer diferentes parámetros como el consumo de CPU o memoria RAM, memory leaks, o el rendimiento OpenGL del terminal. Más información en [15].

Por el otro lado, el SDK de Android incluye diversas herramientas de desarrollo como el AVD Manager, con el que es posible manejar dispositivos virtuales Android, proyectos y componentes del propio SDK, un emulador con el que testear las aplicaciones desde el mismo ordenador (Android Emulator), y un debugger (Dalvik Debug Monitor Server). A diferencia de iOS, este SDK no proporciona un IDE propio, por lo que es necesario utilizar otras opciones como Eclipse o NetBeans. El lenguaje de alto nivel necesario para desarrollar en Android es Java. Más información en [16].

2.3.1.2. Distribución digital

Ambas plataformas poseen los siguientes servicios integrados de distribución digital de aplicaciones accesibles desde un terminal compatible:

- App Store (iOS).
- Android Market.

El App Store se caracteriza por ser la primera tienda digital en un dispositivo móvil, donde todos los usuarios de iOS pueden descargar miles de aplicaciones desde cualquier lugar del planeta. El lanzamiento de este servicio propició la aparición de numerosas aplicaciones creadas por desarrolladores independientes y empresas de software, que vieron como una gran oportunidad de negocio el poder crear y distribuir sus ideas sin necesidad de invertir grandes cantidades de dinero en publicidad. Sus aplicaciones serían visibles por millones de usuarios desde sus smartphones.

Apple, por su parte vio también una gran oportunidad de negocio, ya que permitir a los desarrolladores publicar sus aplicaciones en el App Store le reporta un 40% de los ingresos obteniendo por cada una de ellas, quedando el 60% restante para los desarrolladores. Esto originó numerosas críticas por considerarse un alto precio a pagar, después de todo el esfuerzo humano y económico que conlleva desarrollar una aplicación.

Casi a la par que Apple, Google lanza el Android Market con un objetivo similar al App Store, que cualquier usuario pudiese instalar aplicaciones en su terminal Android de una manera sencilla y económica. Con la intención de superar al App Store, Google rebaja el “impuesto” por publicar una aplicación en el Android Market, de modo que el desarrollador recibe el 70% de los ingresos totales, quedando el 30% para Google.

Aunque el SDK de ambas plataformas sea gratuito, el envío de una aplicación para su publicación en el App Store o Android Market requiere de un pago previo. En el primer caso, es necesario abonar a Apple una cuota anual de 99$, mientras que en el segundo hay que abonar un único pago de 25$.

2.3.1.3. Mercado de aplicaciones

El estudio del comportamiento de las tiendas descritas en el apartado anterior se ha convertido en un tema de vital importancia para desarrolladores y empresas, debido a la enorme expansión que están sufriendo ambas plataformas en los últimos meses (cada día se activan cerca de 900.000[6] dispositivos móviles con sistema operativo Android).

En el mes de Febrero de 2011, la compañía de seguridad *Lookout Mobile Security* publicó los resultados de un informe[7] en el que se detalla el presente y futuro de ambas plataformas, en cuanto a cantidad y precio de aplicaciones, y a número de desarrolladores. Los resultados son los siguientes:

I) *El número de aplicaciones en el Android Market ha crecido un 127% en el último año, por un 44% del App Store;* este último dispone de casi 350.000 aplicaciones, de las cuales, 100.000 han aparecido en los últimos 6 meses. Mientras, Android Market dispone de cerca de 90.000 aplicaciones con 50.000 de ellas aparecidas también en los últimos 6 meses (ver Figura 6).

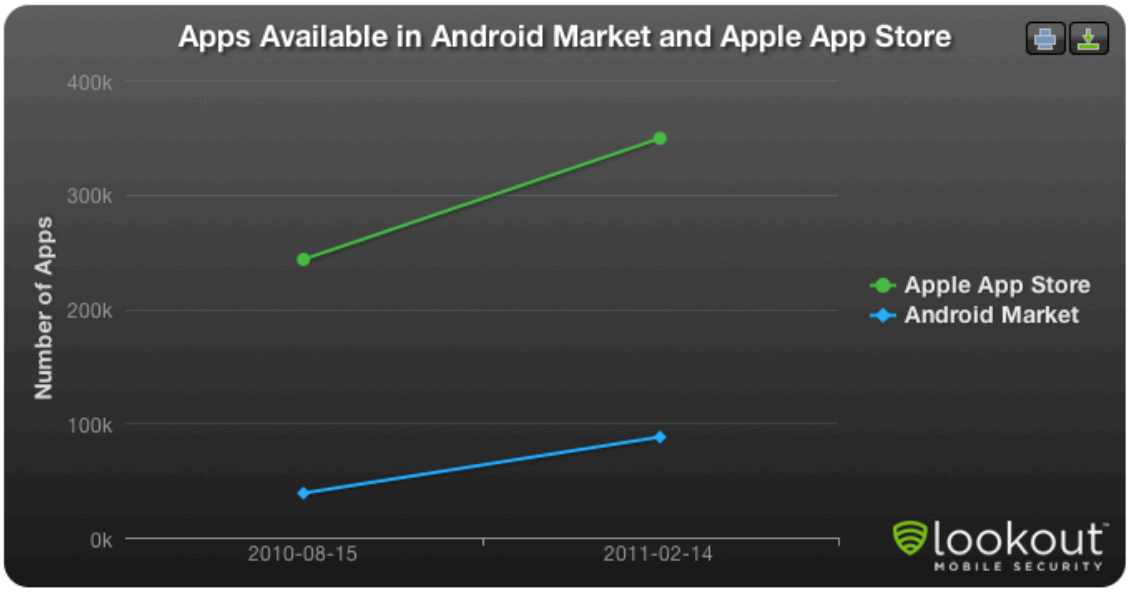

Figura 6: Evolución en el número de aplicaciones disponibles en iOS y Android.

[6] Ver información en [26]

[7] Ver [27]

II) *El número de desarrolladores inscritos en el App Store creció un 48% en los últimos 6 meses, por un 40% en el Android Market (ver Figura 7);* a pesar del mayor incremento en el número de desarrolladores en el App Store, el ratio de aplicaciones publicadas por desarrolladores fue mayor en el Android Market un 6.6 frente al 4.8 del App Store (ver Figura 8).

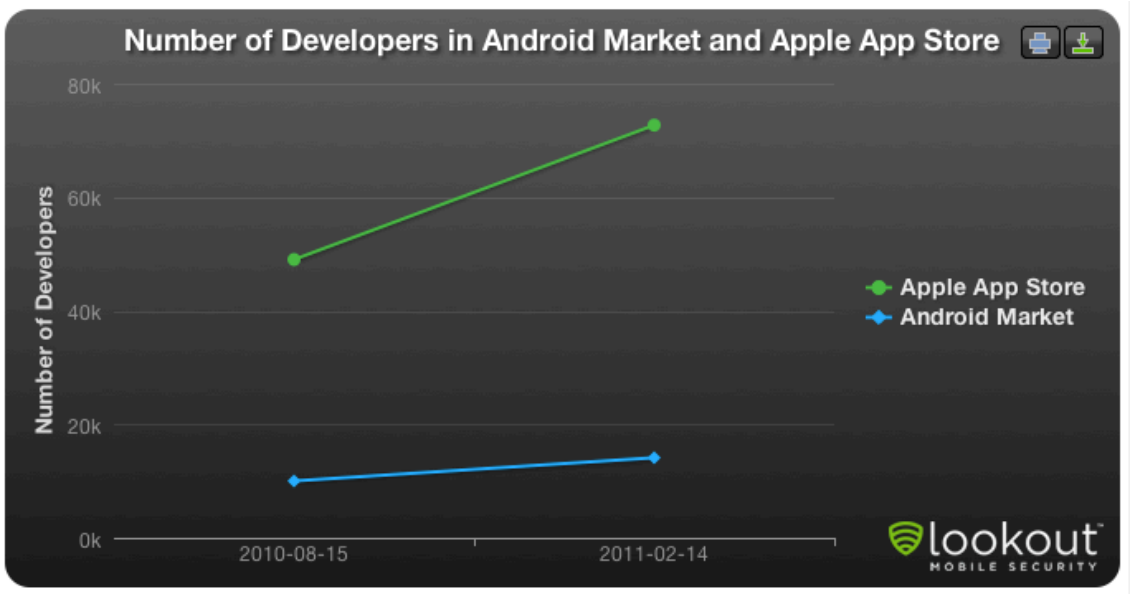

Figura 7: Evolución en el número de desarrolladores inscritos en el App Store y Android Market.

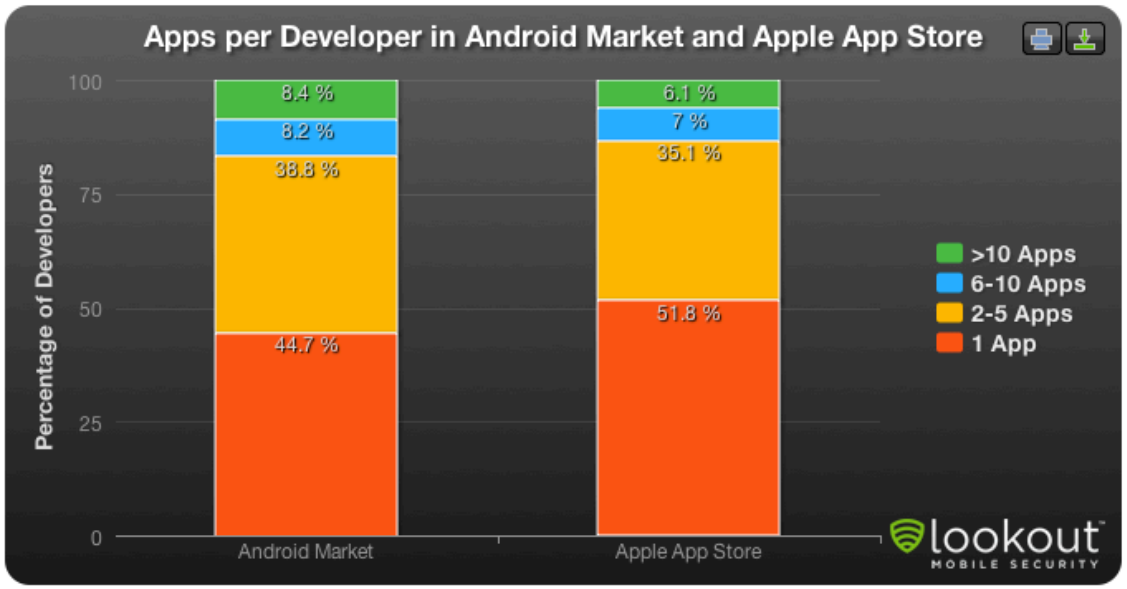

Figura 8: Número de aplicaciones enviadas por desarrollador en el App Store y Android Market.

III) *El porcentaje de aplicaciones de pago creció del 22% al 34% en el Android Market en los últimos seis meses, mientras que en el App Store se redujo del 70% al 66% (ver Figura 9).*

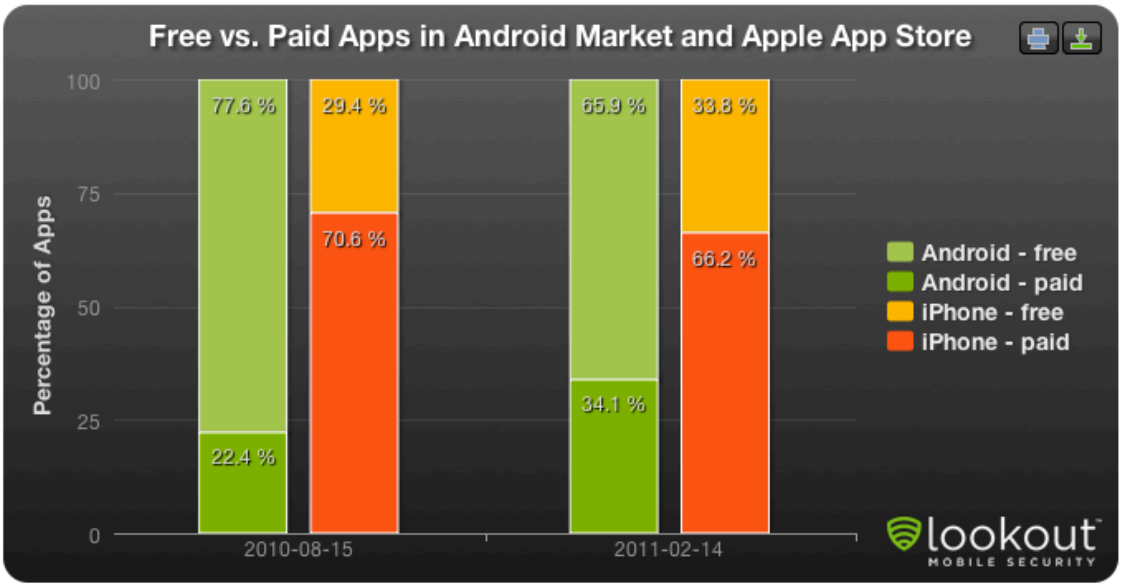

Figura 9: Porcentaje de aplicaciones de pago y gratuitas en el App Store y Android Market.

IV) *El precio de las aplicaciones ha crecido notablemente en el Android Market durante los últimos 6 meses, mientras que en el App Store se ha mantenido prácticamente igual (ver Figuras 10 y 11).*

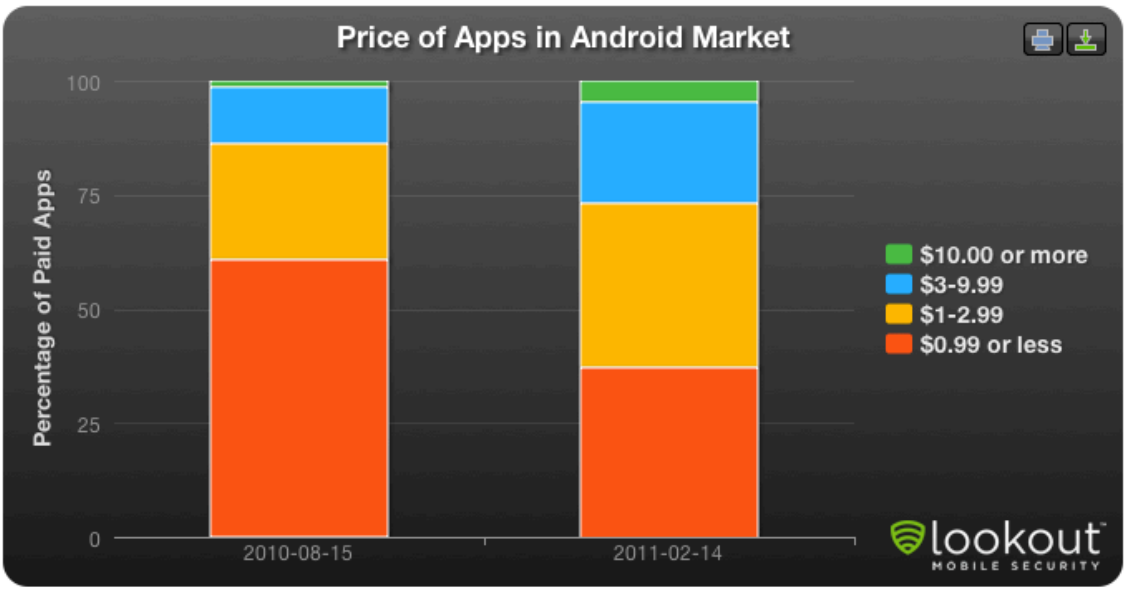

Figura 10: Rango de precios en aplicaciones del Android Market.

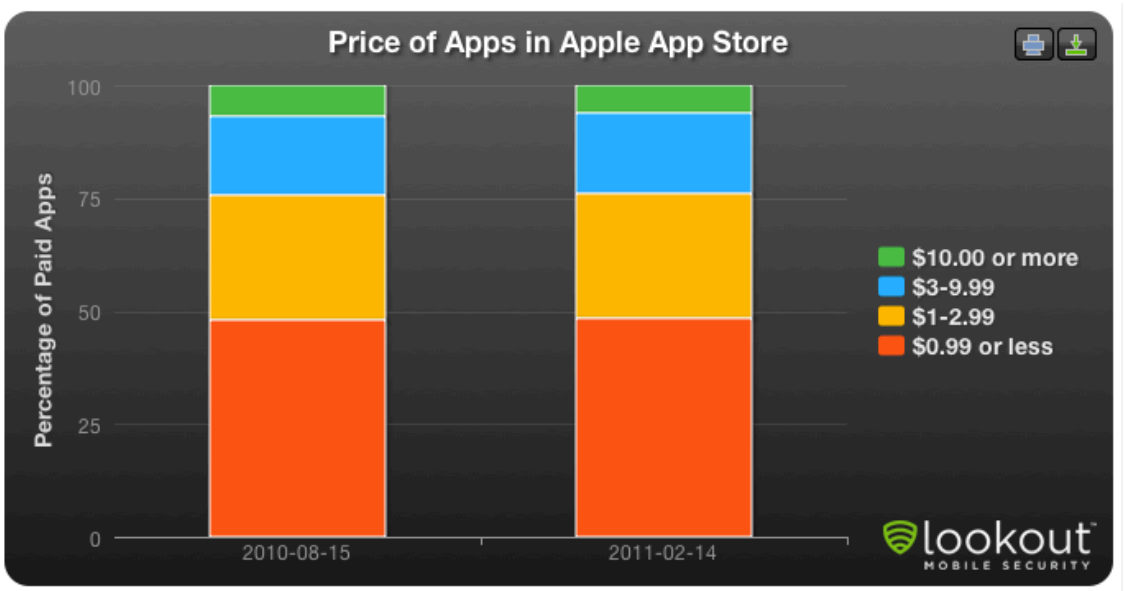

Figura 11: Rango de precios en aplicaciones del App Store.

Todos estos datos indican una cierta ventaja del App Store sobre el Android Market. Por el momento, los desarrolladores prefieren a iOS para crear y vender sus aplicaciones, a pesar de percibir un menor porcentaje de los beneficios que en Android, debiéndose principalmente a lo comentado en apartados anteriores, en los que se muestra la facilidad de desarrollo que ofrece el SDK de Apple frente al de Google. Esto ha provocado un enorme crecimiento en el número de desarrolladores inscritos en el App Store durante los últimos años, sin tampoco olvidar la buena aceptación que está teniendo Google con su sistema operativo móvil Android, como se puede observar en los resultados mostrados con anterioridad.

Notar el alto porcentaje de aplicaciones gratuitas disponibles en el Android Market (casi un 66%). Esto es un indicador claro de que los usuarios de Android no gastan mucha cantidad de dinero en la compra de aplicaciones. Por el otro lado, el porcentaje en el App Store de este tipo de aplicaciones disminuye hasta casi el 34%, lo que hace pensar que una aplicación puede reportar mayores beneficios, pues existe un mercado con millones de usuarios con gran disposición a adquirirla.

Por último, el aumento en el precio que han sufrido las aplicaciones del Android Market da una idea de la necesidad por parte de los desarrolladores de intentar generar más beneficios. Mientras el App Store apenas ha experimentado cambios en los últimos meses al respecto, por lo que parece que los desarrolladores ven correcto el precio de sus aplicaciones y están contentos con el rendimiento económico de las mismas.

2.3.1.4. Puntos débiles

Actualmente, la plataforma Android se distribuye a largo de todo el mundo en una gran variedad de dispositivos móviles, de marcas y modelos muy diferentes entre sí. Esto provoca una fragmentación en el mercado que pone en series dificultades a los

desarrolladores, a la hora de adaptar su aplicación para toda esa diversidad de dispositivos. Una correcta implementación de la aplicación conlleva testearla en todo el hardware existente en el mercado, algo difícil de cumplir para un desarrollador independiente. Esto no ocurre en la plataforma iOS, pues tan sólo existen 3 dispositivos: iPhone, iPod Touch y iPad.

Por otro lado, los desarrolladores de la plataforma iOS deben seguir unas reglas muy estrictas, impuestas por Apple, que rigen el contenido publicado en el App Store. Es por ello, por lo que todo desarrollador que planee publicar su aplicación en dicha tienda, debe tener presente la posibilidad de que ésta puede ser rechazada si no cumple alguna de las normas establecidas en [5]. En este punto, Android ofrece una mayor flexibilidad, otorgando mayor libertad de creación al desarrollador.

2.3.1.5. Conclusión: elección de una tecnología

Tras una detallada descripción de las plataformas móviles iOS y Android, las principales razones por las que se ha escogido la primera de ellas son:

- Facilidad en el desarrollo; el SDK de iOS es tremendamente intuitivo y fácil de usar desde el comienzo. Tan sólo es necesario descargarse el paquete Xcode 4, instalarlo y comenzar a escribir código. Las herramientas de desarrollo ofrecen una serie de funcionalidades que facilitan enormemente el laborioso trabajo de crear una aplicación: diseño de interfaces sin necesidad de escribir código (arrastrar y soltar elementos), testeo y corregido de errores en el código a través de una interfaz gráfica (programa *Instruments*).

- Distribución y venta; a pesar de que la plataforma Android está experimentando un gran auge en cuanto a descargas y número de aplicaciones disponibles se refiere, iOS ofrece un servicio de distribución digital de aplicaciones de calidad (App Store), y en la que muchos desarrolladores confían actualmente[8].

- Preferencia por la plataforma; no sería honesto negar la predilección del autor de este proyecto por la plataforma iOS. Posee el hardware necesario (Mac + iPhone + iPad) para su desarrollo, y un conocimiento previo del lenguaje Objective-C que le anima a crear una aplicación para dicha plataforma. No obstante, la decisión ha sido de manera objetiva y totalmente consensuada por el autor y los tutores del proyecto.

2.3.2. Tecnologías en la capa de servidor

Desde un principio, el servidor se ha posicionado en este proyecto como la arquitectura de software más compleja en diseñar, pues en ella se centralizarán los datos de usuarios finales, empleados y administradores, y se coordinarán todas las peticiones realizadas por éstos desde sus aplicaciones móviles o web. Sin servidor el

[8] Cerca de 75.000 desarrolladores inscritos en el App Store, según [27].

funcionamiento del sistema es inviable. Es por esto, por lo que la tecnología con la que se va a programar debe ser una herramienta eficaz y robusta, y que ofrezca una gran escalabilidad ante posibles cambios o mejoras.

El estudio de este componente se ha centrado en las tecnologías con las que el autor de este proyecto se encontraba más familiarizado: PHP y Ruby on Rails. Se tomó esta decisión pues aprender una nueva tecnología completamente desde cero supondría un retraso importante en el desarrollo y un resultado, tal vez, no del todo óptimo.

Ambas tecnologías son lenguajes de programación del lado servidor, lo que significa que son interpretados y ejecutados por el propio servidor, enviando una respuesta al usuario en un lenguaje leíble por su equipo, como por ejemplo: HTML o JavaScript.

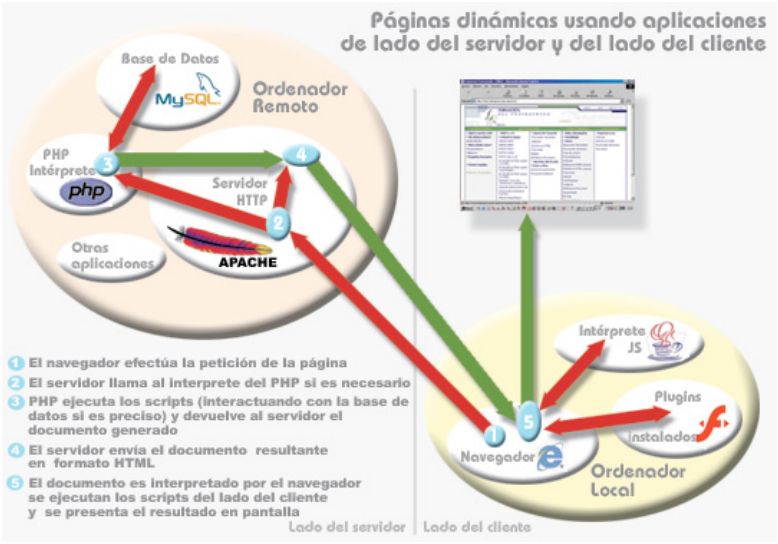

Figura 12: Tecnología de servidor[9].

Las principales características de ambas tecnologías vienen resumidas a continuación.

2.3.2.1. PHP

Es un lenguaje interpretado de alto nivel muy extendido en la programación de páginas web dinámicas, y que se caracteriza por ser de código abierto y embebido en HTML:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>Ejemplo 1</title>
  </head>
  <body>
    <?php
```

[9] Imágen obtenida en [28].

```
        echo "Hola, Mundo!";
    ?>
  </body>
</html>
<!-- El script PHP se incrusta dentro del código html con las etiquetas <?php y ?>  -->
```

Su ejecución se lleva a cabo enteramente en el servidor, enviando al usuario un HTML con los resultados obtenidos, sin que éste pueda conocer como es el código del script PHP.

La característica más importante de este lenguaje es su compatibilidad con una gran cantidad de bases de datos: MySQL, ODBC, PostgreSQL, SQLite. Esto ha provocado que PHP se haya convertido en unos de los lenguajes de programación más extendidos dentro del campo de la programación web, al ofrecer un amplio abanico de posibilidades para establecer una base de datos según las necesidades del desarrollador.

La instalación y configuración de este lenguaje es tremendamente sencilla: tan sólo es necesario un servidor Apache o IIS, con las librerías correspondientes de PHP. Actualmente, los sistemas operativos MAC OS X o LINUX ya lo llevan incluido, por lo que nada más que hay que configurarlo.

Sus ventajas frente a otros lenguajes análogos son:

- Fácil aprendizaje.
- Gran colección de librerías y funciones.
- Gran comunidad de desarrolladores.
- Ejecución rápida.
- Capacidad para conectarse con la mayoría de los tipos de base datos: MySQL, Oracle, SQL Server, etc..
- No es necesario la definición de los tipos de variables.
- Extensa documentación en la página oficial[10].

Las desventajas que presenta son:

- Dificultad en la programación orientada a objetos (es necesario utilizar frameworks externos).
- Modularización complicada.

[10] Más información [29].

2.3.2.2. Ruby on Rails

Es un Framework de código abierto pensado especialmente para el desarrollo de aplicaciones web, cuya base es el lenguaje de programación Ruby. Su principal objetivo es facilitar enormemente el desarrollo de webs dinámicas con poco código y una mínima configuración.

El método de trabajo que ofrece Ruby on Rails se basa en dos principios básicos: "*Don`t Repeat Yourself (DRY)*" y "*Convention Over Configuration*". El primero busca evitar la duplicidad de información que perjudique la legibilidad del código; mientras que el segundo promueve que el desarrollador no tenga que tomar decisiones de más, que en adelante puedan entorpecer el desarrollo general de la aplicación web. Para dar una idea más clara de estos principios se propone el siguiente ejemplo: si el desarrollador crea una clase "*Persona*", su tabla correspondiente se llamará *Personas*. En caso de que no se desee ese nombre, el desarrollador lo indicará con una instrucción:

set_table_name "people"

Al igual que PHP, Ruby on Rails ofrece total soporte para diferentes tipos de bases de datos MySQL, PostgreSQL, Oracle, etc.. El acceso a la información almacenada en cada una de ellas es independiente de su tipología, ya que existen una serie de mecanismos únicos para cualquier clase de base de datos. Sólo es necesario especificar en la configuración del servidor el tipo de base de datos que se desea utilizar.

A la hora de realizar búsquedas dentro de las tablas incluidas en la base de datos, Rails establece un capa de abstracción sobre esta última, simplificando el código enormemente:

```
def getShows

    @posts = Post.find(10).reorder('name');

end

# La instrucción find equivaldría a la siguiente query:
# SELECT * FROM posts WHERE id = 10 ORDER BY name
```

Este Framework confía en el patrón de diseño Modelo - Vista - Controlador (MVC), del todo conocido entre los programadores, y que estructura la programación en tres partes (ver con más detalle en el capítulo 3). En el caso de la vista, Rails permite también empotrar su código dentro del HTML, de modo que, por ejemplo, se pueda mostrar al usuario el valor de una variable definida en el controlador:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<html>
   <head>
      <title>Ejemplo 2</title>
   </head>
   <body>
      <table>
            <% @posts.each do |p| %>
                  <tr>
```

```
                    <td><%=p.id%></td>
                </tr>
            <% end %>
        </table>
    </body>
</html>
<!-- El código de Ruby on Rails se incrusta dentro del código html con las etiquetas <% y <%=  -->
<!-- La variable @posts proviene de la función getShows incluida en el controlador -->
```

Ruby on Rails ofrece numerosas ventajas, como:

- Independencia de la base de datos.
- Soporte para diversos tipos de base de datos.
- Uso del patrón MVC.
- Facilidad de desarrollo.
- Flujo de trabajo más rápido.
- Programación orientada a objetos.
- Gran cantidad de plugins externos [11].

Mientras que sus desventajas son:

- Difícil instalación en ciertos Sistemas Operativos (p.e. Mac OS X).
- Pocos servidores compatibles, en comparación de otros lenguajes de programación del lado servidor.

2.3.2.3. Conclusión: elección de una tecnología

Decidirse por el uso exclusivo de uno de estos lenguajes es una tarea complicada, pues ambos ofrecen características y funcionalidades que pueden facilitar el desarrollo de este proyecto, en algún momento concreto del mismo. Sin embargo, se ha decidido en utilizar el Framework Ruby on Rails para la programación del servidor por las siguientes razones:

- Facilidad en el desarrollo; una de las características más destacables de este Framework es la obtención de primeros resultados en un corto período de tiempo, debido a su sencillez y a su abstracción en un nivel más alto de la base de datos.

- Compatibilidad con el paradigma de arquitectura REST (Representational State Transfer); una de las premisas a tener en cuenta en la implementación del servidor es el modo de adquirir o escribir los datos desde las aplicación móvil. Básicamente, la plataforma iPhone ofrece un excepcional soporte para

[11] Conocidos como "*gems*".

peticiones HTTP mediante los métodos estándar: GET y POST. Es en esto, en lo que la técnica de programación RESTful consiste: obtención y respuesta de recursos XML, JSON, HTML, etc…, mediante un URL a través del protocolo HTTP. Por todo ello, Ruby on Rails permitirá una perfecta comunicación entre el dispositivo del usuario final y el servidor.

- Preferencia por el lenguaje; desde el comienzo, el autor del proyecto y sus tutores pensaron en utilizar una plataforma que facilitase, en la medida de lo posible, el desarrollo sin que perjudicase el resultado final del mismo. Tras estudiar con detenimiento que ofrecían, tanto PHP como Ruby on Rails, este último cumple con creces los requisitos del proyecto.

2.3.3. Escenarios reales

Para un diseño viable y práctico de la aplicación se ha llevado a cabo un estudio de aquellos servicios cotidianos de mayor relevancia, cuya principal característica es la reserva previa de cita. Ello ha servido de base para definir en cierta medida varios requisitos funcionales del sistema pero sobre todo para implantar un proceso de reserva de citas en la aplicación móvil con funcionalidades similares a dichos servicios pero también con mejoras que lo hiciesen más intuitivo.

A continuación aparecerán tres pequeños relatos con un actor llamado Daniel que decide reservar una cita en tres servicios muy habituales de citas previa. El objetivo de éstos es proporcionar una visión general de lo comentado en el párrafo anterior, de modo que el lector de este documento adquiera pequeñas nociones sobre cómo se lleva a cabo habitualmente un proceso de cita previa desde una página web, y pueda hacer una mejor comparación entre dichos servicios y el implementado en este proyecto.

Atención médica primaria

Un día cualquiera, Daniel decide acudir a su centro de salud más cercano a pedir una cita para su médico de cabecera, pero de repente, recuerda que puede hacerlo desde el salón de su casa con tan sólo conectarse a internet en su portátil. Sin más dilación, enciende su equipo, conecta con el portal web de salud de la Comunidad de Madrid, y pincha en el enlace "*Cita previa en Atención Primaria*", encontrándose con esto:

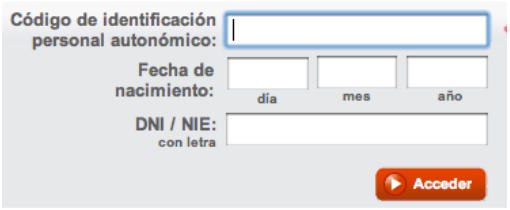

Figura 13. Pantalla de inicio de sesión para la cita previa en atención primaria.

Rápidamente coge su tarjeta sanitaria e introduce el código de identificación que aparece en ella, además de su fecha de nacimiento y de su DNI. Pincha en el botón "*Acceder*". Tras la espera correspondiente en la carga de la página, ve lo siguiente:

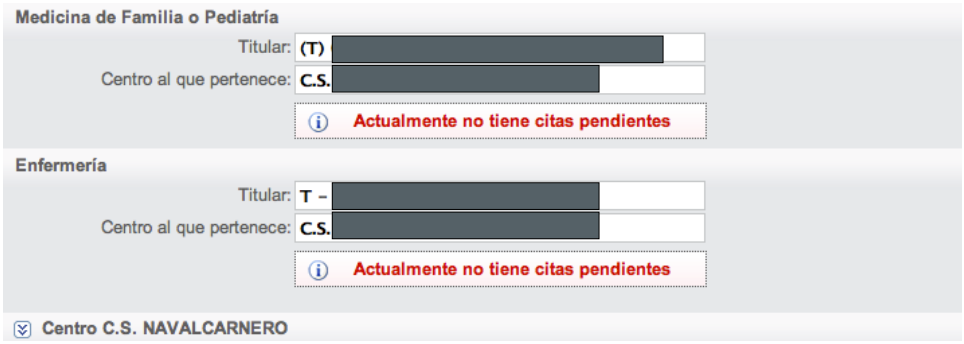

Figura 14. Lista de empleados y centro de salud asignados.

No tiene porqué preocuparse. No tiene ninguna cita pendiente con su médico, ni con su enfermero. Durante unos segundos, busca donde puede pedir una cita, hasta que encuentra el enlace. Clickea, y descubre:

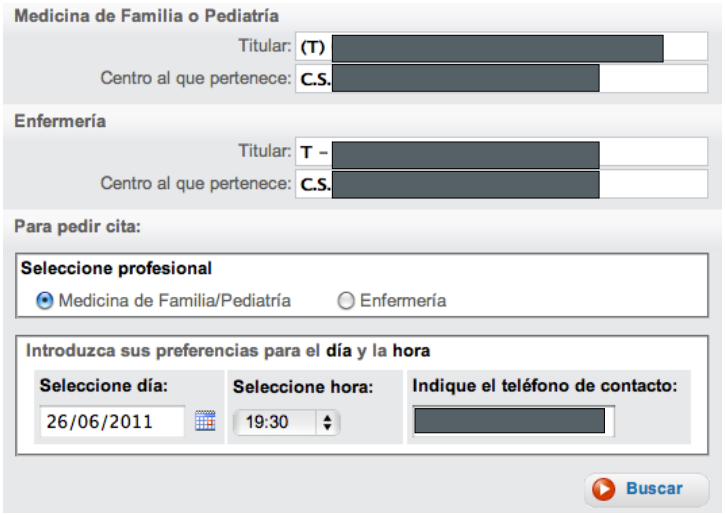

Figura 15. Formulario para la reserva de una cita en atención primaria.

Sin pensarlo más, elige el día y la hora que le vendría mejor, y escribe su número de teléfono. Espera a que el sistema compruebe la disponibilidad de su médico, cuando de repente, recibe el siguiente mensaje:

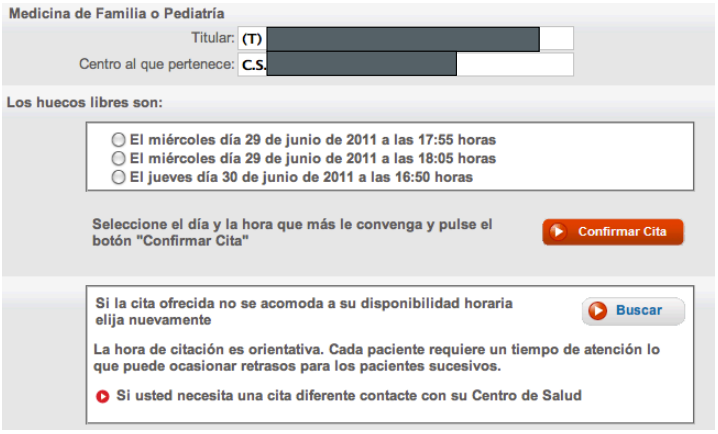

Figura 16. Selección de cita disponible para su reserva en atención primaria.

¡Vaya! esos días le vienen realmente mal. Tras pensarlo durante un rato, recuerda que el jueves está libre. Así que confirma la cita, e imprime un justificante en su impresora por si se olvida y llega tarde.

Un par de días más tarde, su jefe le recuerda la reunión acordada desde la semana anterior para el mismo día y la misma hora que su cita con el médico. No le queda más remedio que cambiarla.

En su hora libre del trabajo, se conecta nuevamente al portal web donde hizo la reserva, y pincha en "*cambiar o anular cita*". La web le ofrece la posibilidad de anularla o de cambiarla, pero él se decide por la última opción, ya que está muy interesado en hablar con su médico. Tan sólo repite de nuevo los mismos pasos que realizó cuando hizo la reserva. Elige un nuevo día y una hora, confirma la reserva, e imprime un nuevo justificante. Ya no tiene que preocuparse más por ello.

Servicio de renovación de DNI

Estando charlando con sus amigos sobre trámites légales en los que hay que presentar el DNI, Daniel recuerda que el suyo está próximo a caducar. Nada más llegar a casa, se conecta a Internet y entra la página del Ministerio del Interior, donde encuentra el enlace para la petición de cita previa. Como en la mayoría de los casos, se encuentra con una pantalla de bienvenida en la que tiene que ingresar los datos de su carné de identidad actual para así, poder ser reconocido por el sistema.

Figura 17. Pantalla de inicio de sesión en el servicio de renovación de DNI.

Tras teclear todos y cada uno de ellos, pincha en "*Enviar datos*", y llega a una nueva página en la que elige el tipo de documento oficial que quiere renovar, que en este caso es su DNI. A continuación, selecciona la comunidad autónoma en la que reside y la comisaría de policía más cercana a su casa. Es entonces, cuando el sistema le ofrece la posibilidad de elegir la fecha y hora que más le conviene.

Figura 18. Selección de citas disponibles para la renovación del DNI.

Viendo las fechas y horas que oferta el sistema, Daniel se decide por el Miércoles 24 a la 13:30, pues a esa hora tendría tiempo para comer con más tranquilidad. Después de hacer la selección, se encuentra con la siguiente pantalla:

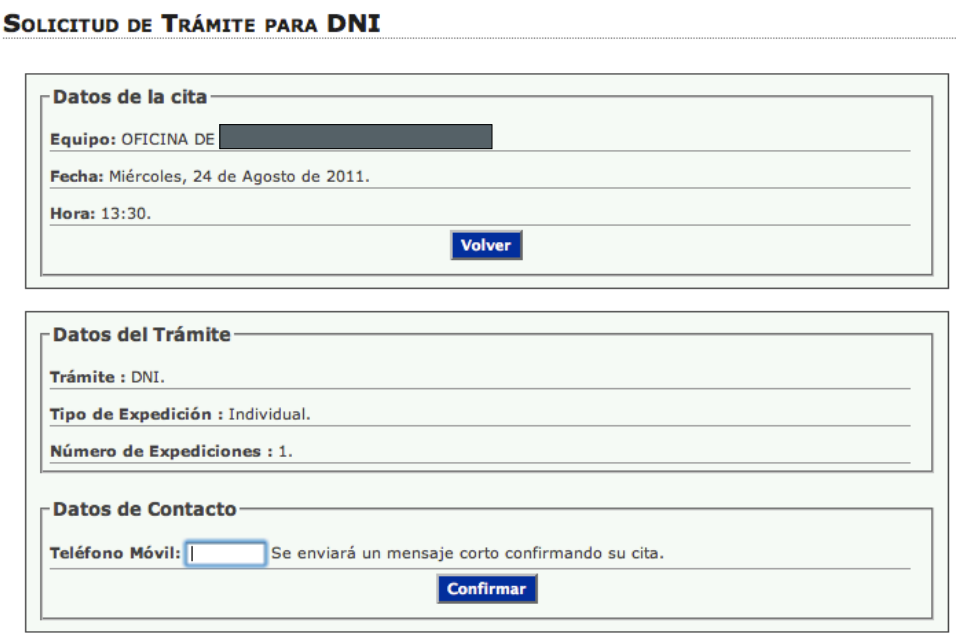

Figura 19. Confirmación de reserva de cita para la renovación del DNI.

Se trata de una pantalla de confirmación, en la que el sistema permite a Daniel introducir su teléfono móvil para poder enviarle un sms con todos los datos referentes a la cita. Sin más dilación, lo introduce y pulsa el botón "*Confirmar*".
Tras un momento de espera, recibe el sms en su teléfono móvil y llega a la siguiente página web:

Figura 20. Vista detalle de la cita reservada y confirmada para la renovación del DNI.

Aquí ve toda la información de la cita, en la que se incluye un número de cita. Con dicho número, Daniel podrá navegar por la aplicación web en caso de tener que realizar otra gestión.

Semanas más tarde, Daniel decide modificar la fecha de la cita por tener un viaje de empresa durante esos días. Accede nuevamente a la aplicación web del Ministerio del Interior e introduce los datos requeridos en la misma. A continuación, entra en la ficha de "Consulta de Citas" y observa que no puede modificar su cita. Necesita primero anularla y después reservar una nueva. Se pone manos a la obra.

Pincha en el enlace "*Anulación de citas*" y se encuentra con una ficha de su cita. Asociada a ella, aparece un botón de anulación. Sin pensarlo, lo pulsa y recibe inmediatamente una confirmación de que su cita ha sido anulada correctamente. Dentro de la propia confirmación aparece un enlace para reserva una nueva cita. Daniel hace click en él y vuelve a la primera pantalla, en la que tiene que seleccionar el tipo de documento oficial que necesita renovar. Lo siguientes pasos a realizar son los mismos que llevó a cabo semanas antes.

Atención al cliente de Apple Inc

En el día de su cumpleaños, Daniel recibe un Macbook Pro como regalo. Durante los primeros días de uso, no entiende algunos aspectos del funcionamiento de su nueva computadora. Por ello, decide reservar una cita en el Apple Store más cercana para que un experto le pueda aconsejar. El sistema de reservas de citas que ofrece Apple Inc. se desarrolla enteramente en su página web, por lo que Daniel sólo tiene que encender su equipo y entrar en ella.

Lo primero que se encuentra es un formulario en el que tiene que introducir su Apple ID y una contraseña. Al ser la primera vez que hace uso de este servicio debe rellenar un formulario con el que conseguirlos.

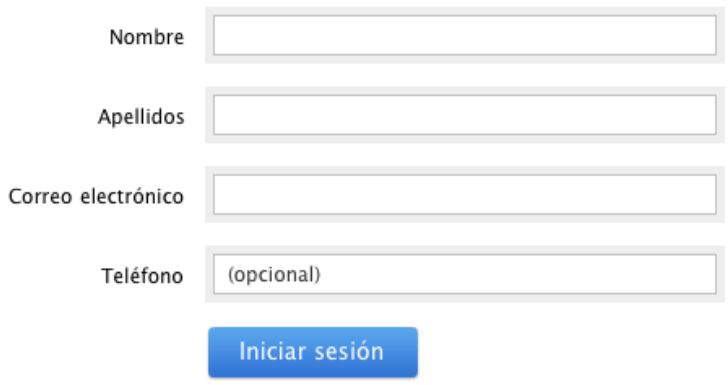

Figura 21. Pantalla de inicio de sesión para la reserva de cita en el servicio de atención al cliente de Apple.

Después de rellenar los campos requeridos y esperar a la confirmación del sistema, ya puede iniciar su sesión. Dentro de ésta, lo primero que se encuentra es una lista con sus citas reservadas, la cuál, en este momento se encuentra vacía. Como su intención es la de tratar con un experto, pincha en el icono de "*Reservas en Genius Bar*" y lleva a una pantalla de selección de dispositivo.

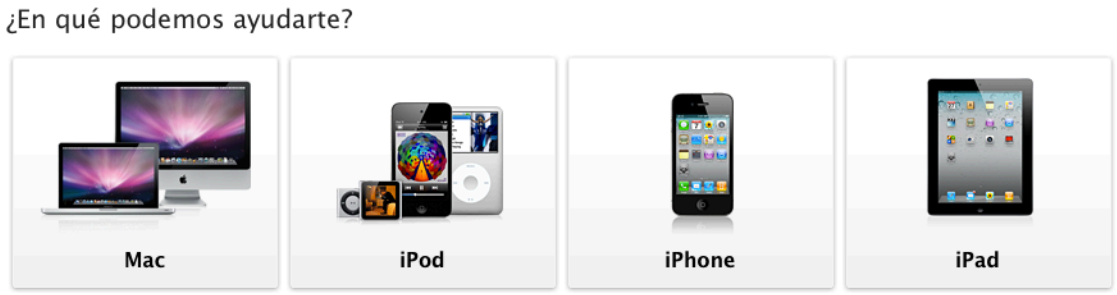

Figura 22. Selección de actividad donde reservar la cita en el servicio de atención al cliente de Apple.

Aquí sin dudarlo, Daniel selecciona la opción de Mac y pasa a una nueva pantalla de elección de horario. El sistema le ofrece unas fechas y unas horas muy próximas, aspecto que le agrada enormemente.

Figura 23. Selección de cita disponible para su reserva en el servicio de atención al cliente de Apple.

Rápidamente, escoge la fecha y la hora que más le conviene y pulsa "*Continuar*". A continuación, llega a la confirmación de su cita.

Figura 24. Vista detalle de confirmación de reserva de una cita para el servicio de atención al cliente de Apple.

En este pantalla, la aplicación ofrece a Daniel varias posibilidades: visualizar un mapa de la localización exacta de la Apple Store, añadir un breve comentario para que el empleado conozca de antemano los motivos de su petición de cita, o incluso ver todas sus citas pendientes. En este caso, se decide por la primera opción, y así comprobar exactamente como llegar a la tienda. Finalmente, pulsa el botón "*Hecho*". Ya tiene su cita reservada.

Unas horas antes de la cita, Daniel no recuerda con exactitud la hora de la misma, así que decide comprobarla en la página web. Inicia su sesión con los mismos datos que ingreso la primera vez, y se dirige a la pantalla de citas reservadas. Comprueba que su cita es a la 13:30, así que todavía tiene tiempo.

Figura 25. Lista de todas las citas reservadas en el servicio de atención al cliente de Apple.

El sistema también le permite cambiar la fecha o cancelarla. En el primero de los casos, tan sólo tendría que elegir un nuevo horario, mientras que en el segundo, con pulsar la opción "Cancelar" ya quedaría anulada la cita por completo.

2.3.4. App Store

Dentro de la plataforma móvil de Apple, existe un servicio de distribución de aplicaciones, denominado App Store, en el que cualquier usuario puede adquirir una aplicación (a elegir entre más de 550.000[12]) de manera cómoda y rápida, accediendo desde cualquier dispositivo iOS, o bien, desde un ordenador personal, ya sea PC o Mac, en el que esté instalado el software iTunes [13] de Apple (ver Figura 26).

[12] Datos obtenidos en [30].

[13] Ver [31].

Figura 26: Acceso al App Store[14].

Su funcionamiento consiste en acceder a la aplicación App Store, integrada en los dispositivos mencionados anteriormente, y navegar entre sus diferentes categorías y listas donde se incluyen las aplicaciones más descargadas, de mayor popularidad o las más recientes. Cuando el usuario ha localizado una aplicación que desea comprar, tan sólo tiene que pulsar el botón *"Comprar app"* y su descarga comenzará inmediatamente. Tras descargarse e instalarse de manera automática, la aplicación esta lista para ser utilizada. Más información en [6] y [7].

Al escoger este medio de distribución, se ha de tener en cuenta que su utilización siempre deberá estar sometida a una serie de normas[15] impuestas por Apple, que homogeneizan el proceso de envío de la aplicación a la propia Apple, y su puesta a la venta en el App Store. Así se asegura que todos los programadores, que deseen vender su aplicación en el App Store, sigan los mismos pasos y tarden el menor tiempo posible.

Actualmente, el App Store contempla dos posibles tipos de aplicaciones: gratuitas y de pago. Las primeras engloban a aquellas que no suponen un desembolso económico para el usuario a la hora de adquirirlas, aunque sí pueden incluir un pago posterior a la compra por habilitación de funcionalidades extras o por descarga de contenidos adicionales[16], además de las que hacen uso de publicidad externa mediante la inclusión de banners (ver apartado 2.3.5). Por el lado contrario, las aplicaciones de pago implican pagar una cierta cantidad de dinero (tabulada por región[17]) para poder ser descargadas. Es en esta topología donde el coste para el programador por publicar su aplicación en el App Store es el 30% de las ganancias totales, cantidad que retiene Apple por ofrecer este canal de distribución. El 70% restante queda íntegro para el desarrollador.

---

[14] Imágenes obtenidas en [31].

[15] Ver [8] y [9].

[16] Ver [32].

[17] Ver [10].

Haciendo un breve resumen del App Store, se puede decir que:

- ✓ Ofrece un método sencillo y cómodo para adquirir aplicaciones.
- ✓ Presenta un catálogo de aplicaciones muy extenso.
- ✓ Impone ciertas reglas a los desarrolladores en los envíos de aplicaciones.
- ✓ Ofrece aplicaciones gratuitas y de pago.
- ✓ Los precios están tabulados.
- ✓ Los desarrolladores reciben el 70% de los beneficios totales, mientras que el 30% restante es para Apple.

2.3.5. Publicidad iAd

En 2010, Apple lanzó el servicio de publicidad móvil iAd con el objetivo de permitir a los desarrolladores incluir una publicidad interactiva en sus aplicaciones de una manera sencilla y adecuada a cualquier interfaz (ver Figura 27).

Antes de que el servicio iAd apareciese en escena, la publicidad en los dispositivos iOS era símbolo de masificación y desorden de los banners, lo cual, disminuía enormemente la usabilidad de las aplicaciones perjudicando la gran experiencia de usuario que ofrece el sistema operativo móvil iOS. Su puesta en marca asegura que la publicidad llegue a cualquier usuario sin interferir en su interactividad con la aplicación, y que el objetivo principal de un anuncio de publicidad, ser perfectamente visible durante el mayor tiempo posible, se cumpla.

Figura 27: Ejemplo de publicidad con banner en el servicio iAd[18].

Con este servicio no sólo se beneficia al usuario final que utiliza la aplicación, sino también al desarrollador. Apple se encarga de suministrar a este último los anuncios que aparecerán embebidos dentro de su aplicación, y el software *iAd Producer*[19] con el

[18] Imagen obtenida en [33].

[19] Ver más información en [11].

que implantarlos dentro de ella. Un ejemplo de ello, es el anuncio interactivo[20] diseñado para el servicio iAd por la empresa automovilística Renault, publicitando uno de sus primeros vehículos eléctricos dentro de una aplicación iOS, y en el que el usuario puede "*jugar*" dentro del anuncio, agitando el dispositivo para cambiar el color de la carrocería, o incluso reservando un punto de recarga para no tener que esperar tras su llegada a la estación de suministro.

De todos estos ingresos generados se reparte el 60% para el desarrollador que incluye publicidad iAd en su aplicación, y el 40% restante para Apple por mantener este servicio. Más información en [12].

Actualmente, este servicio solamente esta disponible en un pocos países (ver Figura 28), aunque se espera que se extienda rápidamente por otros mercados internacionales, incrementando los beneficios enormemente.

Figura 28: Lista de países con disponibilidad para iAd[21].

[20] Vídeo demostración del anuncio: http://www.youtube.com/watch?v=Jv7YXiDHzF4

[21] Imagen obtenido en [12]

CAPÍTULO 3: DISEÑO

En este capítulo se va a tratar de explicar de manera amplia el diseño de la arquitectura empleada en las capas descritas en el capítulo anterior (ver Sección 2.3). Además, se incluirán varios diagramas de secuencia para entender varios procesos de gran importancia durante la ejecución del sistema.

Figura 29. Diseño general del sistema.

3.1. Arquitectura de la capa de servidor

El diseño de la arquitectura para la capa de servidor se ha planteado atendiendo a las necesidades del sistema y a la tecnología utilizada para su implementación (Ruby on Rails).

Ha sido necesario el uso de dos patrones de arquitectura muy comunes gestionar todos los recursos del sistema (REpresentational State Transfer) y para hacer el código lo más legible y modulado posible (Model-View-Controller). Con ellos se ha facilitado enormemente la tarea de desarrollo de todas las aplicaciones involucradas en el sistema.

A continuación se explicarán detalles generales de ambos patrones.

3.1.1. REpresentational State Transfer

Este patrón de arquitectura consiste en la representación del estado de un recurso. Cuando se habla de recurso, se está refiriendo a una entidad como por ejemplo "*cita*". Todos los recursos poseen un identificador único que por lo general se corresponde con un número entero, siendo su representación de estado un formato de datos determinados. Por ejemplo y para que quede un poco más claro, la representación del estado de una cita con identificador *45* puede ser un archivo XML, HTML o incluso JSON.

Su metodología consiste en la utilización de los siguientes verbos incluidos en el protocolo HTTP a través de una URL:

- *GET*; obtención de un recurso.
- *POST*; creación de un recurso.
- *PUT*; actualización de un recurso.
- *DELETE*; eliminación de un recurso.

Cada recurso posee una URL única, de modo que cualquiera de éstos es accesible desde una URL. Si se quisiese, por ejemplo, acceder a un empleado con identificador 12 sólo sería necesario lanzar una petición HTTP con el método GET a la siguiente URL: http://www.citapreviauc3m.es/employee/12. Su representación se definiría en la cabecera "*Accept*" de la petición como "*application/json*" o "*application/xml*" si es necesaria en format JSON o XML respectivamente.

Como se puede observar, una de las grandes ventajas que ofrece esta arquitectura es la elección del formato de la representación del recurso. REST puede devolver la información en XML o en JSON.

XML (Extensible Markup Lenguage) es un lenguaje de etiquetas, creado por la World Wide Web Consortium (W3C), cuyo propósito es establecer un estándar para el intercambio de información entre diferentes lenguajes de programación y plataformas[22]. El problema que presenta este lenguaje es la cantidad de recursos que se consume al parsear su contenido. Aquí es donde surge el formato JSON para intentar solucionar dicho problema.

JSON (JavaScript Object Notation) es un formato de texto para un intercambio más ligero de información[23]. Además es independiente de cualquier lenguaje y plataforma, y se caracteriza por usar convenciones regulares simulares a lenguajes de programación muy extendidos. El uso de este formato en este proyecto prevalecerá por encima de XML, pues uno de los objetivos es usar el mínimo ancho de banda en la comunicación entre capas, debido a las limitaciones de la red 3G (reducción en la velocidad de descarga tras consumir una cantidad de bytes al mes, problemas de cobertura,….) presentes en los dispositivos móviles iOS.

[22] Más información en [18]

[23] Más información en [19]

3.1.2. Model-View-Controller

El patrón de arquitectura de software Model-View-Controller, conocido como MVC, implica separar la interfaz gráfica de la lógica que se esconde detrás de ellas. Esto permite a los desarrolladores modelar ambas partes de manera totalmente independiente, facilitando su reutilización y mantenimiento.

Su implantación se reduce básicamente a una estructuración del código en tres componentes:

- **Modelo;** contiene todo lo relacionado con el acceso a las base de datos y el tratamiento de la información.
- **Vista;** como bien dice su nombre, en este bloque se integra todo lo que el usuario va a visualizar en la pantalla (botones, tablas, campos de texto,…).
- **Controlador;** es el encargado de enlazar los bloques anteriores, procesando la interacción del usuario en la vista, de modo que el modelo proporcione la información a mostrar, o bien, la modifique.

En el caso del Framework Ruby on Rails, se puede decir que éste cumple perfectamente con dicho patrón. En la Figura 30, se observa la estructuración que realiza Rails cuando es creada una nueva aplicación. Éste automáticamente genera una serie de rutas en las que depositar los ficheros de código que pertenecen al controlador (1), modelo (2) y a la vista (3). En los dos primeros, la extensión de los archivos es *rb* (código puramente Ruby), mientras que en el último es *html.erb*. Esto se debe a que en la vista el código predominante es HTML, permitiendo incrustar porciones de código Ruby cuando es necesario acceder a atributos o métodos definidos en en el controlador.

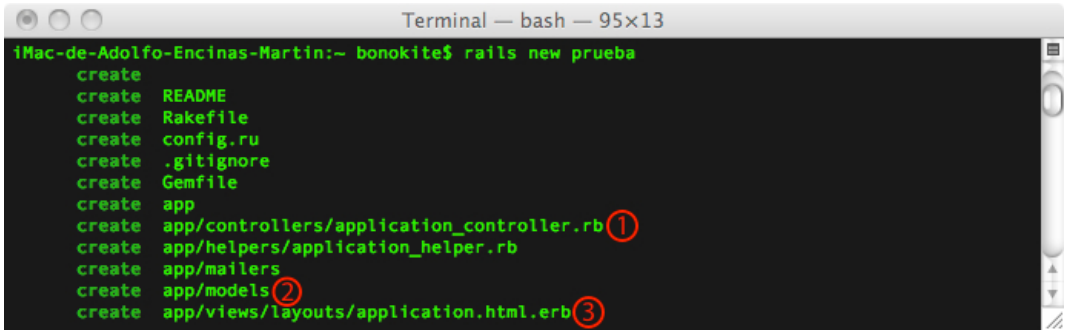

Figura 30. MVC en Ruby on Rails.

3.1.3. Componentes

Una vez definidos los patrones de arquitectura de software utilizados en el diseño de la capa de servidor, es necesario ver de qué manera han sido aplicados (ver Figura 31).

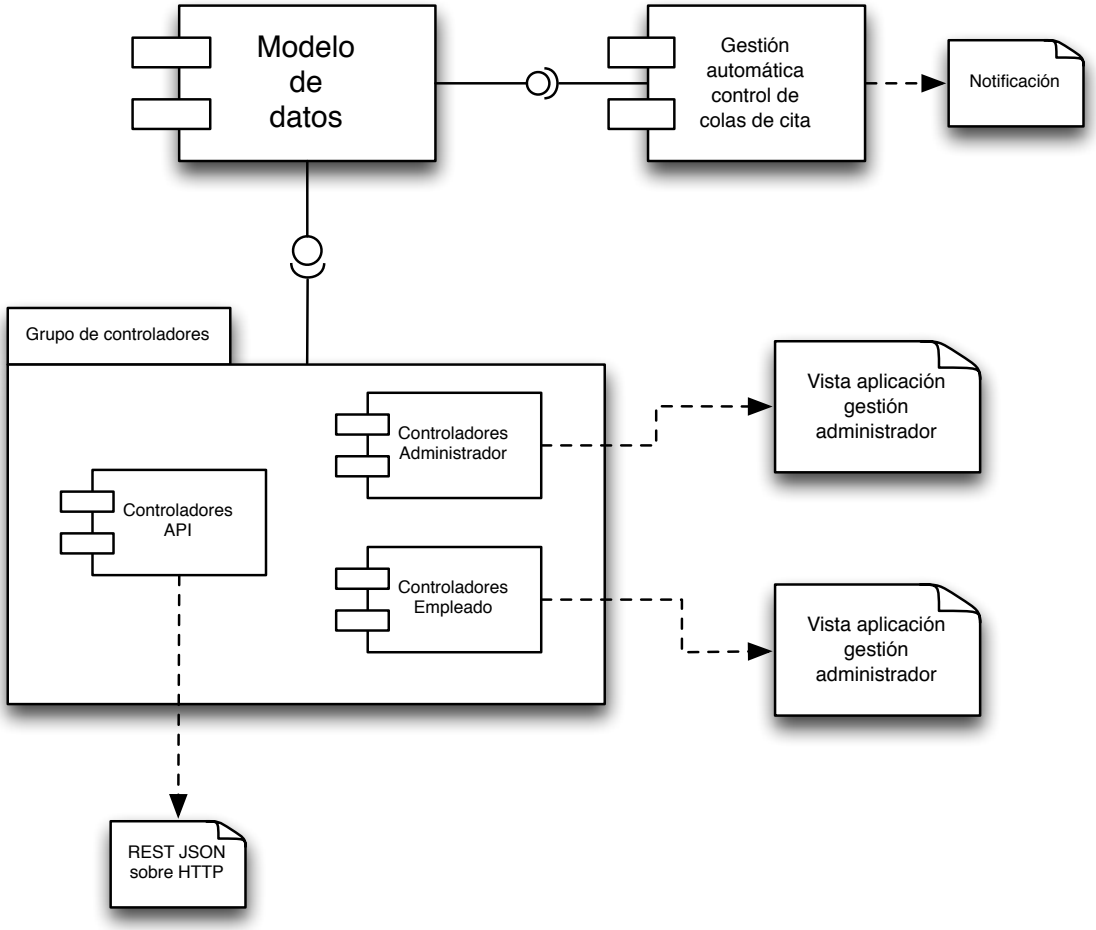

Figura 31. Componentes en la arquitectura de la capa de servidor.

Es fácil comprobar como el patrón MVC está presente en la arquitectura del servidor. Se tiene un modelo de datos, un grupo de controladores y un grupo de vistas.

Con respecto al patrón REST, éste aparece como un componente que parte de los controladores que conforman la API móvil, y que representa el estado de uno o varios objetos del modelo en formato JSON para que puedan ser leídos por un dispositivo móvil. Asimismo, las dos vistas presentes para las aplicaciones de gestión también forman parte de dicha arquitectura. Equivalen al componente anterior pero esta vez, en vez de representar un estado en formato JSON, lo hacen en formato HTML.

Notar la existencia de un componente encargado de gestionar el automatizado del estado sobre las colas de citas. No se trata de ningún controlador, modelo o vista. Puede considerarse como un módulo que toma del modelo los datos necesarios para

realizar el cálculo del estado de cada cola, y el posterior envío del resultado a los usuarios correspondientes por medio de notificaciones a su dispositivo móvil.

En los siguientes apartados se pasará a describir los componentes más relevantes de la Figura 31. Se comenzará con el modelo de datos, a continuación vendrán los grupos de controladores, y las vistas para terminar con la gestión automática del control de colas.

3.1.3.1. Modelo de datos

El primer y más importante componente dentro de la arquitectura MVC utilizada en la capa de servidor es el modelo de datos. Es en este componente donde se establecen las entidades necesarias para que el sistema funcione correctamente en su totalidad. Al hablar de entidad, se refiere a una unidad mayor que comprende una tabla en la base de datos y una relación con las demás. Por ejemplo, una entidad puede ser "*Usuario*", y su tabla correspondiente en la base de datos es "*usuarios".* Un objeto[24] de esta entidad puede estar relacionado con varios objetos de la entidad "*Cita",* ya que un usuario final puede tener varias citas reservadas.

Una sobrecarga de entidades o una incorrecta distribución de relaciones entre ellas en el modelo de datos puede provocar que el sistema se vuelva lento e inestable cuando el volumen de consultas y escrituras en la base de datos sea muy elevado. Es necesario que este sea lo más óptimo posible, de modo que el acceso a un registro y a sus relaciones no suponga recorrer demasiadas tablas haciendo que el tiempo de consulta se vea incrementado considerablemente.

Tras un intenso estudio de las posibles entidades que conformasen el modelo de datos se ha llegado a:

[24] Un objeto de entidad está representado como un registro en una fila de la tabla correspondiente.

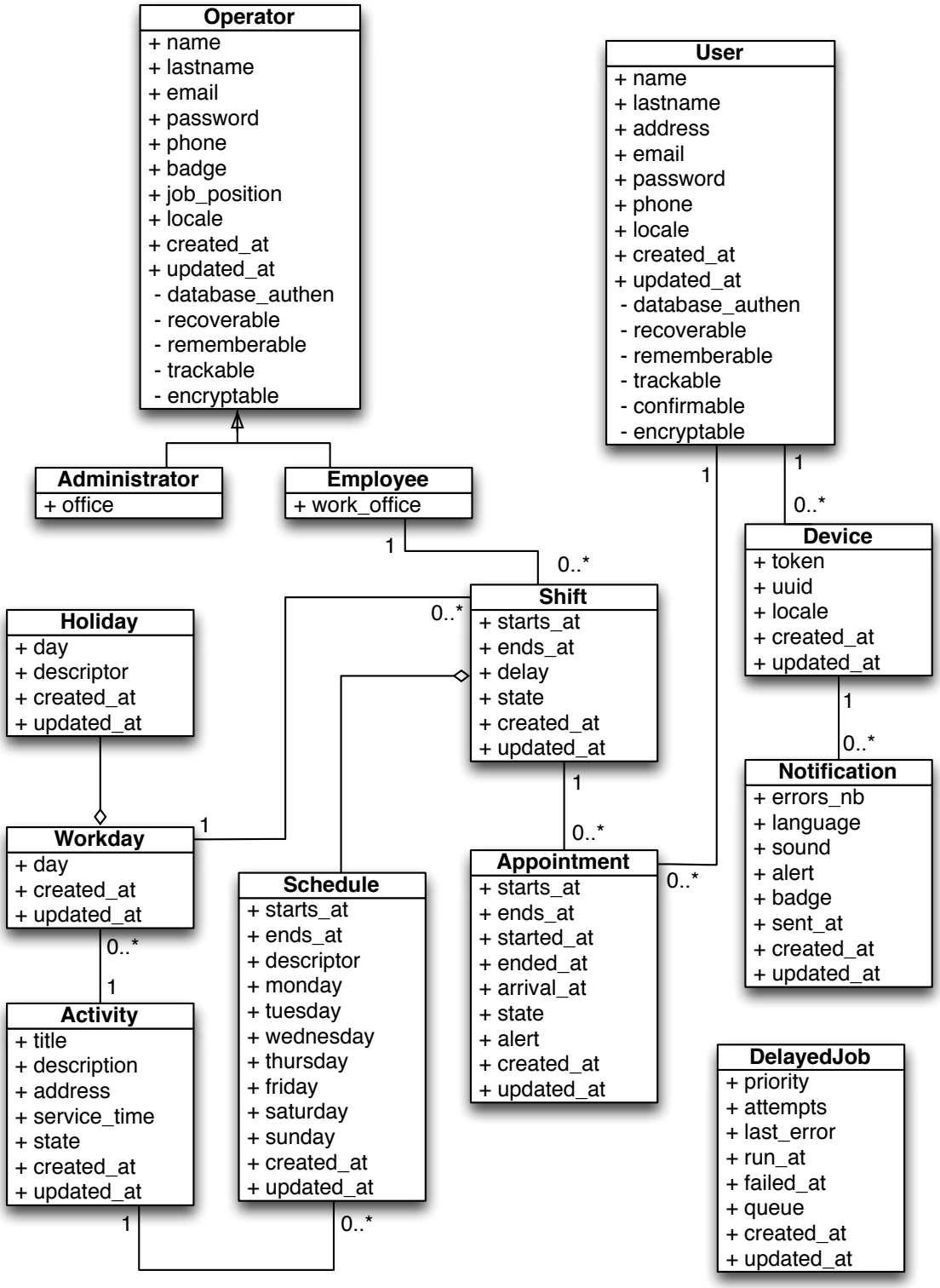

Figura 32. Modelo de datos.

En la Figura 32, se pueden observar las entidades empleadas en el modelo de datos del sistema, junto a sus atributos y relaciones.

En la siguiente lista se detalla la finalidad de cada entidad:

- **Operator:** Se corresponde con el perfil de operador que aparece en la Figura 3. Da origen a una tabla de nombre "*operators"* dentro de la base de datos.

- **Administrator/Employee:** Estas dos entidades representan a los perfiles de administrador y empleado en la Figura 3. Ambas heredan de la clase *Operator* por lo que sus registros se encuentran en la tabla "*operators*", diferenciándose exclusivamente por el atributo *type*.

- **User:** Representa a un usuario dentro del sistema que esté contenido en la tabla "*users"*.

- **Device:** Se corresponde con un dispositivo móvil almacena en la tabla correspondiente de nombre "*devices"*, cuyos registros son creados cada vez que un usuario inicia una sesión desde un dispositivo móvil.

- **Appointment:** Es la entidad básica del sistema correspondiéndose con cada una de las citas que gestiona éste. El nombre de su tabla correspondiente es "*appointments"*.

- **Notification:** Se corresponde con las notificaciones enviadas a los dispositivos móviles de los usuarios, a través del servicio Push de Apple. Su tabla correspondiente en la base de datos, se denomina "*notifications"*.

- **DelayedJob:** Representa una tarea a ejecutar en un momento posterior a una acción determinada. El nombre de la tabla correspondiente es "*delayed_Jobs"*.

- **Holiday:** Simboliza los días de vacaciones en los que el sistema no ofrece ninguna cita para su reserva. La tabla correspondiente se denota como "*holidays".*

- **Activity:** Representa cada una de las actividades o servicios que puede ofrecer el cliente. Su tabla correspondiente en la base de datos es "*activities"*.

- **Schedule:** Se corresponde con los horarios que puede presentar una actividad. Su tabla es "*schedules"*.

- **Workday:** Es cada uno de los días de trabajo programados en cada actividad. Se representa en la base de datos mediante la tabla "*workdays"*.

- **Shift:** Representa un turno completo de trabajo de un empleado para cada actividad. Su tabla correspondiente es "*shifts"*.

Cada entidad aquí descrita posee atributos característicos de su clase. Notar que éstos no son más que los campos o columnas que conforman las tablas, y son:

Operator	
Atributos	
type	Tipo de operador: administrador o empleado.
name	Nombre del operador
lastname	Apellidos del operador
email	Correo electrónico del operador.
password	Contraseña del operador.
phone	Teléfono del operador.
badge	Cadena alfanumérica única en conjunción con el correo electrónico.
job_position	Cargo del operador.
locale	Idioma del operador. Valor por defecto ":es".
created_at	Fecha de creación del operador.
updated_at	Fecha de la última modificación en un atributo.
database_authen	Autenticación habilitada.
recoverable	Restauración de contraseña habilitada.
rememberable	Cookies habilitados.
trackable	Seguimiento de IP habilitado
encryptable	Cifrado de contraseña habilitada.

Tabla 2. Entidad Operator.

Administrator	
Atributos	
office	Oficina del administrador.

Tabla 3. Entidad Administrator.

Employee	
Atributos	
work_station	Puesto de trabajo del empleado

Tabla 4. Entidad Employee.

User	
Atributos	
name	Nombre del usuario.
lastname	Apellidos del usuario.
email	Correo electrónico del usuario. Valor único.
password	Contraseña del usuario.
phone	Teléfono del usuario.
address	Dirección del usuario.
locale	Idioma del usuario. Valor por defecto “:es”.
created_at	Fecha de creación del usuario.
updated_at	Fecha de la última modificación en un atributo.
database_authen	Autenticación habilitada.
recoverable	Restauración de contraseña habilitada.
rememberable	Cookies habilitados.
confirmable	Confirmación por e-mail habilitada.
trackable	Seguimiento de IP habilitado
encryptable	Cifrado de contraseña habilitada.

Tabla 5. Entidad User.

Device	
Atributos	
token	Cadena alfanumérica distintiva del dispositivo móvil utilizada por el servidor APNS (Apple Push Notifications Server) para direccionar las notificaciones correctamente.
uuid	Cadena alfanumérica única para cada dispositivo móvil.
locale	Idioma establecido en el dispositivo móvil.
created_at	Fecha de creación del dispositivo.
updated_at	Fecha de la última modificación en un atributo.

Tabla 6. Device.

Notification	
Atributos	
errors_nb	Registro de errores en el envío de la notificación.
language	Idioma de la notificación.
sound	Sonido de la notificación.
alert	Texto incluido en la notificación.
badge	Número que debe aparecer en la esquina superior derecha del icono de la aplicación cuando la notificación es recibida en el dispositivo móvil.
send_at	Fecha de envío.
created_at	Fecha de creación de la notificación.
updated_at	Fecha de la última modificación en un atributo.

Tabla 7. Entidad Notification.

DelayedJob	
Atributos	
priority	Prioridad de la tarea en el planificador.
attempts	Número de intentos hasta la correcta ejecución.
last_error	Registro del último error en la ejecución.
run_at	Fecha de la ejecución.
failed_at	Fecha del último intento fallido de ejecución.
queue	Identificador de la cola de tareas programadas.
created_at	Fecha de creación de la tarea.
updated_at	Fecha de la última modificación en un atributo.

Tabla 8. Entidad DelayedJob.

Holiday	
Atributos	
day	Fecha del día festivo.
descriptor	Descripción del evento.
created_at	Fecha de creación del día festivo.
updated_at	Fecha de la última modificación en un atributo.

Tabla 9. Entidad Holiday.

Activity	
Atributos	
title	Título de la actividad.
description	Breve descripción sobre la actividad.
address	Dirección donde tiene lugar la actividad.
service_time	Tiempo medio de servicio por cita en la actividad.
state	Estado de la actividad. Puede tomar dos valores: 0 (no se permiten reservas de citas) y 1 (sí se permiten reservas de citas).

Activity	
Atributos	
created_at	Fecha de creación de la actividad.
updated_at	Fecha de la última modificación en un atributo.

Tabla 10. Entidad Activity.

Schedule	
Atributos	
starts_at	Hora de comienzo del horario.
ends_at	Hora de finalizado del horario.
descriptor	Breve descripción del horario.
monday	Valor binario indicando sí el horario es efectivo los lunes.
tuesday	Valor binario indicando sí el horario es efectivo los martes.
wednesday	Valor binario indicando sí el horario es efectivo los miércoles.
thursday	Valor binario indicando sí el horario es efectivo los jueves.
friday	Valor binario indicando sí el horario es efectivo los viernes.
saturday	Valor binario indicando sí el horario es efectivo los sábados.
sunday	Valor binario indicando sí el horario es efectivo los domingos.
created_at	Fecha de creación de la horario.
updated_at	Fecha de la última modificación en un atributo.

Tabla 11. Entidad Schedule.

Workday	
Atributos	
day	Fecha del día de trabajo.
created_at	Fecha de creación del día de trabajo.
updated_at	Fecha de la última modificación en un atributo.

Tabla 12. Entidad Workday.

Shift	
Atributos	
starts_at	Hora de comienzo del turno de trabajo.
ends_at	Hora de finalizado del turno de trabajo.
delay	Adelanto o retraso en minutos del turno de trabajo cuando ésta siendo ejecutado por un empleado. El valor de este atributo será el que se comunique a los usuarios con aquellas citas reservadas dentro del turno de trabajo.
state	Estado del turno de trabajo. Puede tomar los valores: 0 (sin comenzar), 1 (en progreso), 2 (finalizado).
created_at	Fecha de creación del turno de trabajo.
updated_at	Fecha de la última modificación en un atributo.

Tabla 13. Entidad Shift.

En cuanto a las relaciones presentes entre las diversas entidades, cabe destacar que todas mantienen la siguiente estructura:

Entidad A (1) <------> (0..*) Entidad B

Esto implica que un registro de la entidad A puede estar relacionado con varios registros de la entidad B, mientras que un registro de la entidad B solamente puede estar relacionado con otro registro de la entidad A. Sustituyendo nombres de entidades del modelo de datos de la Figura 32 por entidad A y B quedaría:

"Un usuario tiene cero, una o varias citas reservadas, y una cita puede estar reservada únicamente por un usuario."

El objetivo de utilizar este tipo de relación no es más que simplificar las consultas a la base de datos de modo que cuando se requiera un registro de un usuario, el acceso a sus citas reservadas se genere de la forma más rápida y eficaz posible, disminuyendo el tiempo de respuesta del servidor.

3.1.3.2. Grupos de controladores

En la capa de servidor se han establecido tres grandes grupos de controladores, que son:

- **Controladores de administrador;** agrupa una serie de controladores encargados de manejar la aplicación web de gestión destinadas a los posibles perfiles de administradores del cliente.

- **Controladores de empleado;** agrupa a aquellos controladores destinados al control de la aplicación web de gestión para los empleados del cliente.

- **Controladores de API móvil;** agrupa a un conjunto de controladores destinados a permitir la comunicación entre las diferentes capas, es decir, entre la aplicación móvil de servicio y el servidor.

A continuación se pasará a describir cada grupo de controladores individualmente.

3.1.3.2.1. Controladores de administrador

La elección de controladores destinados a dirigir la interacción de un administrador con su correspondiente aplicación web de servicio viene determinada por aquellos casos de uso (ver Figura 3) relacionados con dicho perfil de usuario, y por la entidad dentro del modelo de datos que vayan a gestionar. Por ejemplo, para aquellos casos de uso relacionados con la gestión de usuarios finales (añadir o eliminar) se ha creado un controlador específico, denominado "*UsersControllers*".

En la Figura 33 se detalla mediante un diagrama de clases todos los controladores existentes en la capa de servidor pertenecientes a la aplicación web de gestión para administradores.

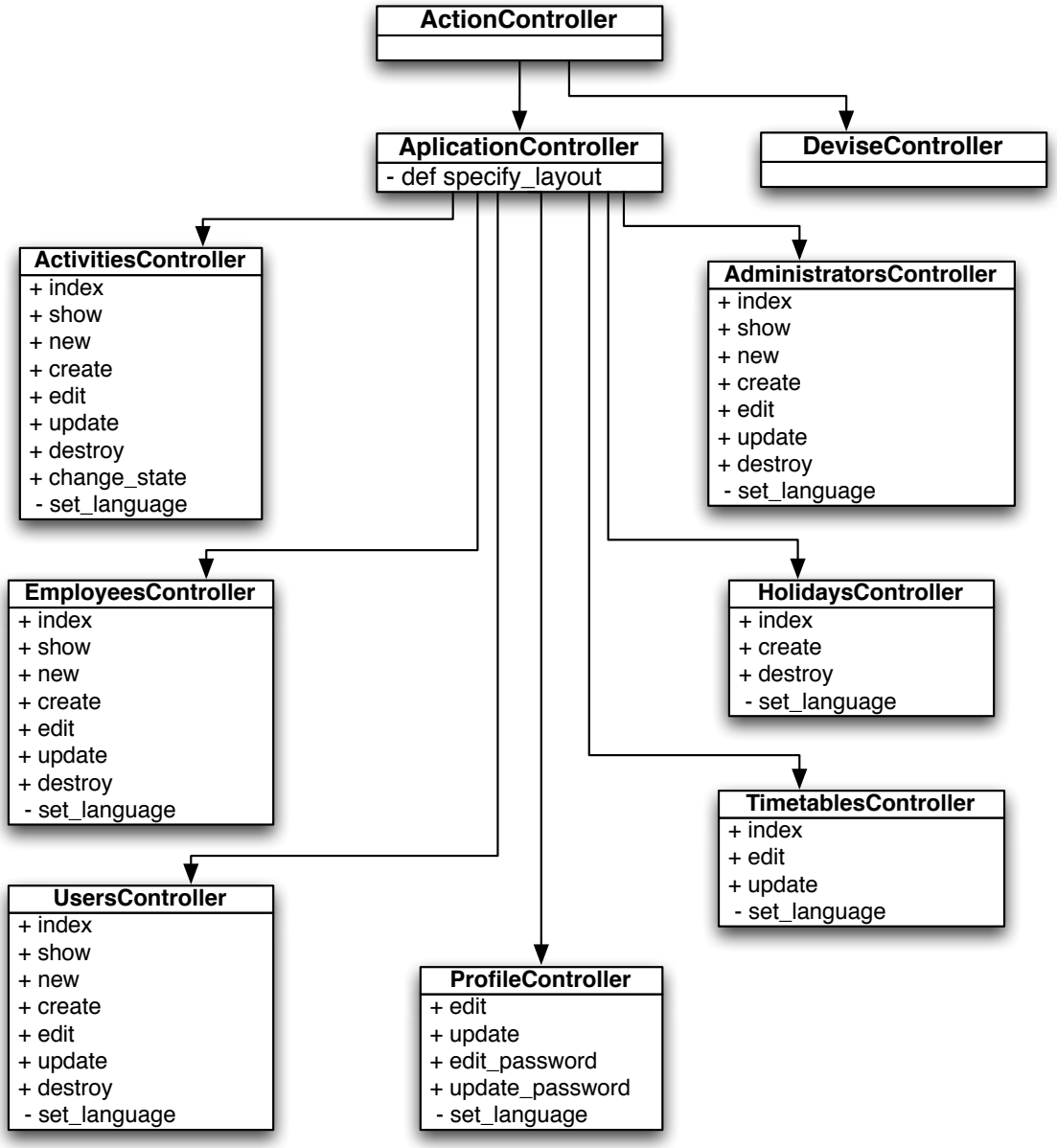

Figura 33. Diagrama de clases: controladores de administrador.

La finalidad de cada una de las clases es:

- **ActionController:** Proporciona diferentes métodos característicos propios del Framework para el direccionamiento de rutas, el renderizado de vistas o el propio control de la seguridad en la aplicación.

- **ApplicationController:** Controlador generado automáticamente al crear cualquier aplicación y del que heredan todos los controladores restantes. En este caso, se utiliza para especificar el esquema de vista HTML según el perfil de usuario que esté realizando la petición.

- **DeviseController:** Permite la gestión de la sesiones de usuario dependiendo de su perfil. Al igual que los dos controladores anteriores, es común para los distintos bloques de controladores.

- **ActivitiesController:** Gestión de las actividades del cliente. Permite a un perfil de administrador: listar, crear, modificar y borrar distintas actividades.

- **AdministratorsController:** Gestión de los diferentes perfiles de administrador. Proporciona a cada uno de éstos diversas operaciones tales como: dar de alta o baja nuevos administradores, o modificar datos personales de aquellos ya existentes.

- **EmployeesController:** Gestión de los diferentes perfiles de empleado por parte de un administrador. Permite añadir o eliminar registros, y modificar datos personales de aquellos ya existentes.

- **HolidaysController**: Proporciona a un administrador los métodos necesarios para gestionar un calendario de días festivos común para todos los empleados. Las acciones permitidas en este controlador son las de visualización, creación y borrado.

- **UsersController:** Proporciona los métodos necesarios para la gestión de los diferentes perfiles de usuario dentro del sistema por parte de un administrador.

- **TimetablesController:** Agrupa aquellas acciones a realizar por un administrador, en torno a la gestión de los turnos de trabajo de cada empleado, y que conforman las diferentes colas de citas disponibles para los usuarios.

- **ProfileController:** Modificación por parte de un administrador de sus datos personales y contraseña.

Mientras que sus métodos son:

ApplicationController	
Métodos	
specify_layout	Selecciona la plantilla HTML adecuada, dependiendo de si la petición proviene de la aplicación web de servicio para un administrador o un empleado.

Tabla 14. Controladores de administrador: ApplicationController.

ActivitiesController	
Métodos	
index	Busca todas las actividades registradas en la base de datos y renderiza la vista correspondiente con el listado de todas ellas.
show	Busca una actividad en la base de datos mediante su ID y renderiza la vista correspondiente con el detalle de toda su información.
new	Genera un objeto temporal de la clase *Activity* y renderiza el formulario necesario para su posterior guardado.
create	Añade una nueva actividad en la base de datos con los parámetros provenientes del formulario renderizado en el método *new*. Redirecciona al método *show.*
edit	Busca una actividad en la base de datos mediante su ID y renderiza el formulario necesario para su modificación.
update	Busca una actividad en la base de datos mediante su ID y actualiza sus campos con los parámetros provenientes del formulario renderizado en el método *edit*. Redirecciona la petición al método *show*.
destroy	Borra una actividad de la base de datos. Redirecciona la petición al método *index*.
change_state	Alterna el estado de una actividad entre los valores 0 (bloqueado) y 1(activo).
set_language	Carga en la vista el lenguaje elegido por el administrador.

Tabla 15. Controladores de administrador: ActivitiesController.

AdministratorsController	
Métodos	
index	Busca todos las administradores registrados en la base de datos y renderiza la vista correspondiente con su listado.

AdministratorsController	
Métodos	
show	Busca un administrador en la base de datos mediante su ID y renderiza la vista correspondiente con el detalle de toda su información.
new	Genera un objeto temporal de la clase *Administrator* y renderiza el formulario necesario para su posterior guardado.
create	Añade un nuevo administrador en la base de datos con los parámetros provenientes del formulario renderizado en el método *new*. Redirecciona al método *show.*
edit	Busca un administrador en la base de datos mediante su ID y renderiza el formulario necesario para su modificación.
update	Busca un administrador en la base de datos mediante su ID y actualiza sus campos con los parámetros provenientes del formulario renderizado en el método *edit*. Redirecciona la petición al método *show*.
destroy	Borra un administrador de la base de datos. Redirecciona la petición al método *index*.
set_language	Carga en la vista el lenguaje elegido por el administrador.

Tabla 16. Controladores de administrador: AdministratorsController.

EmployeesController	
Métodos	
index	Busca todos las empleados registrados en la base de datos y renderiza la vista correspondiente con su listado.
show	Busca un empleado en la base de datos mediante su ID y renderiza la vista correspondiente con el detalle de toda su información.
new	Genera un objeto temporal de la clase *Employee* y renderiza el formulario necesario para su posterior guardado.

EmployeesController	
Métodos	
create	Añade un nuevo empleado en la base de datos con los parámetros provenientes del formulario renderizado en el método *new*. Redirecciona al método *show.*
edit	Busca un empleado en la base de datos mediante su ID y renderiza el formulario necesario para su modificación.
update	Busca un empleado en la base de datos mediante su ID y actualiza sus campos con los parámetros provenientes del formulario renderizado en el método *edit*. Redirecciona la petición al método *show*.
destroy	Borra un empleado de la base de datos. Redirecciona la petición al método *index*.
set_language	Carga en la vista el lenguaje elegido por el administrador.

Tabla 17. Controladores de administrador: EmployeesController.

HolidaysController	
Métodos	
index	Busca todos los días festivos registrados en la base de datos y genera un objeto temporal de la clase *Holiday.* Renderiza la vista correspondiente con el listado de festivos en forma de calendario junto al formulario necesario para la creación de uno nuevo.
create	Añade un nuevo día festivo en la base de datos con los parámetros provenientes del formulario renderizado en el método *index*. Redirecciona al método *index.*
destroy	Borra un día festivo de la base de datos. Redirecciona la petición al método *index*.
set_language	Carga en la vista el lenguaje elegido por el administrador.

Tabla 18. Controladores de administrador: HolidaysController.

UsersController	
Métodos	
index	Busca todos los usuarios registrados en la base de datos y renderiza la vista correspondiente con su listado.
show	Busca un usuario en la base de datos mediante su ID y renderiza la vista correspondiente con el detalle de toda su información.
new	Genera un objeto temporal de la clase *User* y renderiza el formulario necesario para su posterior guardado.
create	Añade un nuevo usuario en la base de datos con los parámetros provenientes del formulario renderizado en el método *new*. Redirecciona al método *show.*
edit	Busca un usuario en la base de datos mediante su ID y renderiza el formulario necesario para su modificación.
update	Busca un usuario en la base de datos mediante su ID y actualiza sus campos con los parámetros provenientes del formulario renderizado en el método *edit*. Redirecciona la petición al método *show*.
destroy	Borra un usuario de la base de datos. Redirecciona la petición al método *index*.
set_language	Carga en la vista el lenguaje elegido por el administrador.

Tabla 19. Controladores de administrador: UsersController.

TimetablesController	
Métodos	
index	Busca todos los turnos de trabajo por empleado y actividad. Renderiza la vista correspondiente con su listado en forma de calendario diario.
edit	Busca los turnos de trabajo de cada empleado para una actividad, grabados en la base de datos. Genera nuevos objetos temporales de clase Shift para los horarios sin turno asignado. Renderiza el formulario correspondiente a la creación de nuevos turnos o al borrado de aquellos ya existentes en forma de calendario semanal.

TimetablesController	
Métodos	
update	Guarda en la base de datos los nuevos turnos de trabajo marcados en el formulario del método *edit* y destruye aquellos deseleccionados en dicho formulario. Redirecciona al método *index*.
set_language	Carga en la vista el lenguaje elegido por el administrador.

Tabla 20. Controladores de administrador: TimetablesController.

ProfileController	
Métodos	
edit	Busca el registro en la base de datos del administrador que ha iniciado sesión y renderiza el formulario necesario para la modificación de sus datos personales.
update	Actualiza los campos del administrador logueado en la aplicación con los parámetros introducidos en el formulario del método *edit*. Renderiza nuevamente el formulario para nuevas modificaciones.
edit_password	Busca el registro en la base de datos del administrador que ha iniciado sesión y renderiza el formulario necesario para la modificación de la contraseña.
update_password	Actualiza el campo contraseña del administrador logueado en la aplicación con los parámetros introducidos en el formulario del método *edit_password*. Renderiza nuevamente el formulario para un nuevo cambio de contraseña.
set_language	Carga en la vista el lenguaje elegido por el administrador.

Tabla 21. Controladores de administrador: ProfileController.

Es notable la repetición de los métodos: *index*, *show*, *new*, *create*, *edit*, *update* y *destroy*. Esto es así, por la técnica de diseño CRUD (create, read, update, delete) adoptada por Ruby on Rails.[25]

[25] Ver más detalles en [24]

3.1.3.2.2. Controladores de empleado

Este grupo de controladores parte de la misma idea que el anterior salvo que en esta ocasión se dirige a la aplicación web de gestión para los distintos perfiles de empleado que pueda presentar el cliente. Sus controladores son definidos igualmente por dos condiciones: casos de uso de los empleados y la entidad a manejar.

Todos los controladores de empleado se muestran en el siguiente diagrama de clases:

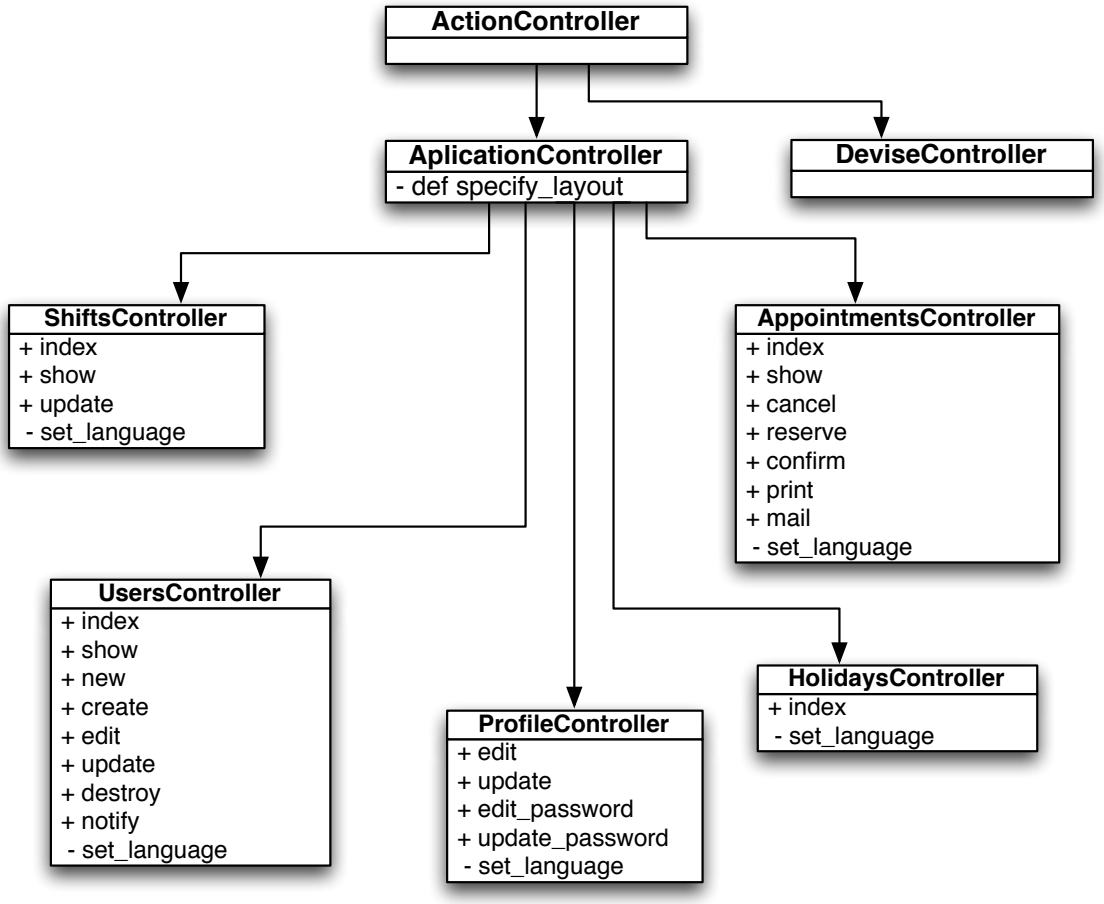

Figura 34. Diagrama de clases: controladores de empleado.

Las clases *ActionController*, *ApplicationController* y *DeviseController* presentan el mismo comportamiento que en el apartado anterior, por lo que no es necesario en este momento y en adelante explicar su finalidad nuevamente. La definición del resto de clases es la siguiente:

- **ShiftsController:** Es el controlador encargado de manejar los turnos de trabajo, y por tanto las colas de citas, del empleado que hace uso de la aplicación.

- **AppointmentsController:** Engloba la gestión por parte del empleado de todas las citas registradas en el sistema.

- **UsersController:** Permite al empleado gestionar cada uno de los perfiles de usuario registrados en el sistema.
- **HolidaysController:** Permite al empleado comprobar los días festivos establecidos por un administrador.
- **ProfileController:** Es el controlador necesario para que el empleado pueda modificar sus datos personales y contraseña en el sistema.

La descripción de los diversos métodos incluidos en cada controlador es:

ShiftsController	
Métodos	
index	Busca todos los turnos de trabajo del empleado desde el día actual y renderiza la vista correspondiente con su listado.
show	Busca el turno de trabajo del empleado para el día de hoy con todas las citas reservadas e inscritas en dicho turno y renderiza la vista correspondiente con su listado.
update	Actualiza el estado del turno de trabajo mostrado en el método *show*, cuando una de sus citas haya comenzado o finalizado, sirviendo de referencia para conocer si éste presenta adelanto o retraso. Renderiza los cambios en la propia vista del método anterior.
set_language	Carga en la vista el lenguaje elegido por el empleado.

Tabla 22. Controladores de empleado: ShiftsController.

AppointmentsController	
Métodos	
index	Busca todos las citas registradas en la base de datos y renderiza la vista correspondiente con su listado.
show	Busca una cita en la base de datos mediante su ID y renderiza la vista correspondiente con el detalle de toda su información.

AppointmentsController	
Métodos	
cancel	Busca una cita en la base de datos mediante su ID y la cancela, pudiendo dejarla o no como disponible para otro usuario según orden del empleado. Renderiza el cambio en la misma vista manejada por el método anterior.
reserve	Busca una cita en la base de datos mediante su ID y cambia su estado a reservada añadiendo la referencia de un usuario. Renderiza la vista correspondiente con los detalles de la reserva.
confirm	Busca una cita en la base de datos mediante su ID y la confirma siempre y cuando su estado previo sea reservada. Renderiza la vista con el mensaje de confirmación.
print	Busca una cita en la base de datos mediante su ID y renderiza los valores de sus campos en una página PDF con posibilidad de impresión.
mail	Busca una cita en la base de datos mediante su ID y envía un E-mail con los valores de sus campos a la dirección de correo personal del usuario que confirmó su reserva.
set_language	Carga en la vista el lenguaje elegido por el empleado.

Tabla 23. Controladores de empleado: AppointmentsController.

UsersController	
Métodos	
index	Busca todos los usuarios registrados en la base de datos y renderiza la vista correspondiente con su listado.
show	Busca un usuario en la base de datos mediante su ID y renderiza la vista correspondiente con el detalle de toda su información.
new	Genera un objeto temporal de la clase *User* y renderiza el formulario necesario para su posterior guardado.

UsersController	
Métodos	
create	Añade un nuevo usuario en la base de datos con los parámetros provenientes del formulario renderizado en el método *new*. Redirecciona al método *show.*
edit	Busca un usuario en la base de datos mediante su ID y renderiza el formulario necesario para la modificación de sus datos personales.
update	Busca un usuario en la base de datos mediante su ID y actualiza sus campos con los parámetros provenientes del formulario renderizado en el método *edit*. Redirecciona la petición al método *show*.
destroy	Borra un usuario de la base de datos. Redirecciona la petición al método *index*.
notify	Busca un usuario en la base de datos mediante su ID y le envía un mensaje escrito por el empleado a través de un formulario incluido en la vista manejada por el método *show*. El mensaje es enviado a los dispositivos móviles del usuario y que se encuentran registrados en la base de datos. Muestra un mensaje con el resultado del envío.
set_language	Carga en la vista el lenguaje elegido por el empleado.

Tabla 24. Controladores de empleado: UsersController.

HolidaysController	
Métodos	
index	Busca todos los días festivos inscritos en la base de datos y renderiza en la vista correspondiente el resultado en forma de calendario mensual.
set_language	Carga en la vista el lenguaje elegido por el empleado.

Tabla 25. Controladores de empleado: HolidaysController.

ProfileController	
Métodos	
edit	Busca el registro en la base de datos del empleado que ha iniciado sesión y renderiza el formulario necesario para la modificación de sus datos personales.
update	Actualiza los campos del empleado logueado en la aplicación con los parámetros introducidos en el formulario del método *edit*. Renderiza nuevamente el formulario para nuevas modificaciones.
edit_password	Busca el registro en la base de datos del empleado logueado y renderiza el formulario necesario para la modificación de la contraseña.
update_password	Actualiza el campo contraseña del empleado logueado en la aplicación con los parámetros introducidos en el formulario del método *edit_password*. Renderiza nuevamente el formulario para un nuevo cambio de contraseña.
set_language	Carga en la vista el lenguaje elegido por el empleado.

Tabla 26. Controladores de empleado: ProfileController.

La mayoría de estos métodos presentan la misma finalidad que los descritos en el bloque de controladores del administrador, pues los perfiles de administrador y empleado comparten varios casos de uso al actuar como un operador (ver Figura 3).

3.1.3.2.3. Controladores de API móvil

El objetivo de este grupo de controladores es el permitir que la aplicación móvil de servicio para usuarios finales se comunique con el servidor, de manera que ésta pueda leer y escribir registros en las tablas de la base de datos. Su función es la de recibir las peticiones realizadas desde una aplicación móvil, ejecutar los métodos necesarios en el modelo (añadir un nuevo usuario final, confirmación de una cita), y devolver una respuesta adecuada, en este caso, en formato JSON. Para ello, ha sido necesario la creación de los siguientes controladores:

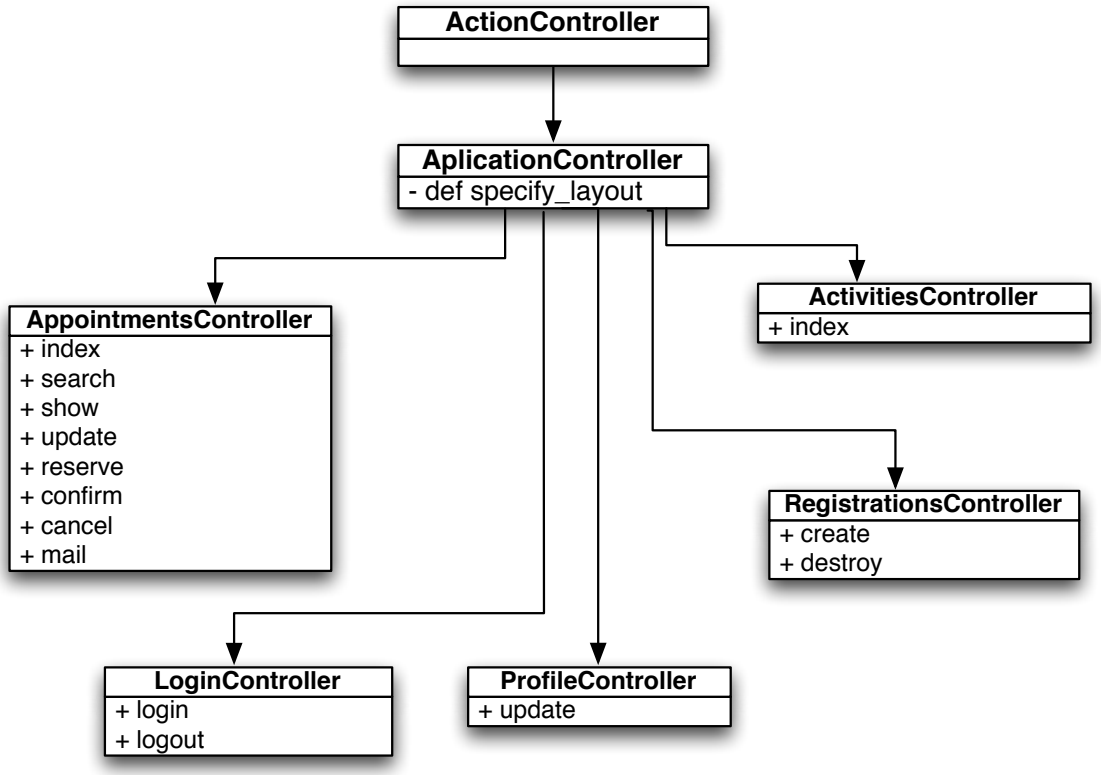

Figura 35. Diagrama de clases: controladores de API móvil.

Cada controlador está destinado a gestionar una entidad diferentes en el modelo de datos:

- **RegistrationsController;** la función de este controlador es permitir a un usuario final crear o destruir una cuenta con la que poder iniciar una sesión privada en el sistema.

- **ProfileController;** controla la actualización de los datos personales del usuario final cuando éste modifica alguno desde su aplicación móvil.

- **AppointmentsController;** este controlador se encarga de gestionar acciones (búsqueda, confirmación, cancelación…) para los registros de citas en la base de datos.

- **LoginController;** se encarga de validar la información de un usuario final permitiendo su entrada al sistema (inicio de sesión), y de asociar el device_token y uuid[26] con su cuenta. Cuando el usuario final cierra una sesión, se encarga de borrar dicha asociación.

- **ActivitiesController;** recoge todas las actividades inscritas en el sistema.

[26] Ver apartado 3.1.3.1

Los métodos de cada controlador son los siguientes:

RegistrationsController	
Métodos	
create	Almacena los datos de un nuevo usuario final en la base de datos y envía un e-mail de confirmación a la cuenta personal de éste.
destroy	Destruye la cuenta de un usuario final registrado en el sistema, borrando toda su información de la base de datos.

Tabla 27. Controladores de API móvil: RegistrationsController.

ProfileController	
Métodos	
update	Actualiza los datos personales de un usuario en la base de datos.

Tabla 28. Controladores de API móvil: ProfileController.

AppointmentsController	
Métodos	
index	Lista todas las citas reservadas por el usuario final.
search	Busca en la base de datos aquellas citas sin reservar en una actividad determinada.
show	Busca una cita reservada por el usuario final para la visualización de toda su información en pantalla.
update	Actualiza el atributo "*alert*" de una cita reservada por el usuario final alternado su valor entre *true* o *false.*
reserve	Cambia el atributo "*state*" de una cita disponible del valor entero 1 a 2 y la asocia al usuario final que está reclamando su reserva. Con esto se consigue la reserva de la cita por parte del usuario final.
confirm	Confirma la reserva de una cita para el usuario final que lo está pidiendo desde su aplicación móvil. Cambia el atributo "*state*" de la cita del valor entero 2 a 3.

AppointmentsController	
Métodos	
cancel	Cancela la reserva de una cita cuando el usuario final lo requiere desde su aplicación móvil. Cambia el atributo *"state"* de la cita del valor entero 3 a 1.
mail	Envía un e-mail con la información de una cita cuando el usuario final que la ha reservado lo requiere desde su aplicación móvil.

Tabla 29. Controladores de API móvil: AppointmentsController.

LoginController	
Métodos	
login	Comprueba las credenciales de un usuario que intenta iniciar una sesión desde la aplicación móvil. Si son correctas permite el acceso y además asocia varias cadenas alfanuméricas identificadoras del dispositivo (*device_token* y *uuid*) al registro del usuario final.
logout	El usuario cierra la sesión destruyéndose la asociación de las cadenas alfanuméricas almacenas en el método anterior con el registro del usuario final.

Tabla 30. Controladores de API móvil: LoginController.

ActivitiesController	
Métodos	
index	Lista todas las actividades disponibles en el servicio que ofrece el cliente.

Tabla 31. Controladores de API móvil: ActivitiesController.

3.1.3.3. Vistas

Sobre el componente de las vistas no hay mucho que decir. El patrón de arquitectura MVC en Ruby on Rails propone que por cada método existente en un controlador haya una vista siempre que sea necesario. De este modo, para un método llamado *index* dentro del *ActivitiesController* será necesario crear un archivo *index.html.erb* en el que se escriba el código encargado de representar de manera visual el resultado de la ejecución de dicho método.

3.2. Arquitectura en la capa de aplicación

Al igual que todo el apartado anterior, la arquitectura empleada para la capa de aplicación estará basada en aquella impuesta por la tecnología elegida durante la etapa de análisis (iOS), y que es Model-View-Controller.

En este punto, no sería necesario mencionar mucho más sobre la arquitectura MVC pues ya fue explicada en el Apartado 3.1.2. No obstante, sí cabría mencionar que la plataforma iOS aplica un tratamiento un poco diferente sobre dicha arquitectura.

En la Figura 36 se muestra un ejemplo de como es aplicado el patrón MVC a una pequeña aplicación cuya función es la de convertirse en una agenda de contactos.

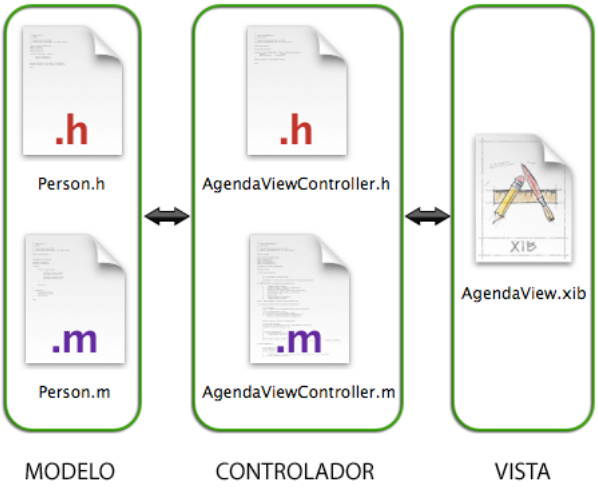

Figura 36. Ejemplo MVC en iOS.

El modelo se encuentra formado por una única entidad, denominada "*Person*", que representa a cada contacto almacenado en la agenda. Su implantación consiste en la conjunción de un archivo de cabecera (*person.h*) donde definir sus atributos y métodos, y otro de implementación (*person.m*) donde desarrollar estos últimos.

La vista se concentra en archivos de interfaz con extensión *xib* (*AgendaView.xib*), en los que el código no es visible, de modo que la configuración y posición de cada uno de los elementos a mostrar en pantalla se realiza en un interfaz gráfica de usuario, valga la redundancia, llamado *Interface Builder*[27].

Y por último, el controlador se encuentra, al igual que el modelo, formado por un archivo de cabecera y otro de implementación (*AgendaViewController.h* y *AgendaViewController.m*, respectivamente). En ellos, se establecen y se escriben las acciones que involucran a la vista con el modelo y sus atributos.

[27] En versiones de Xcode 4 o superiores, Interface Builder ya viene completamente integrado dentro del mismo.

Hasta aquí se puede decir que el patrón MVC en iOS es igual que en Rails. Sin embargo, existe una práctica muy frecuente entre los desarrolladores de iOS, en declarar y escribir elementos visuales pertenecientes a la vista del patrón dentro de los propios controladores. Esto ocurre porque a veces surge la necesidad de personalizar por ejemplo una tabla o un botón, lo cual, supone una tarea prácticamente imposible desde un archivo de interfaz. En este caso, se estaría "infringiendo" de cierta manera el patrón de arquitectura MVC.

A continuación, se muestra el detalle de la arquitectura de la aplicación móvil a alto nivel mediante un diagrama de componentes.

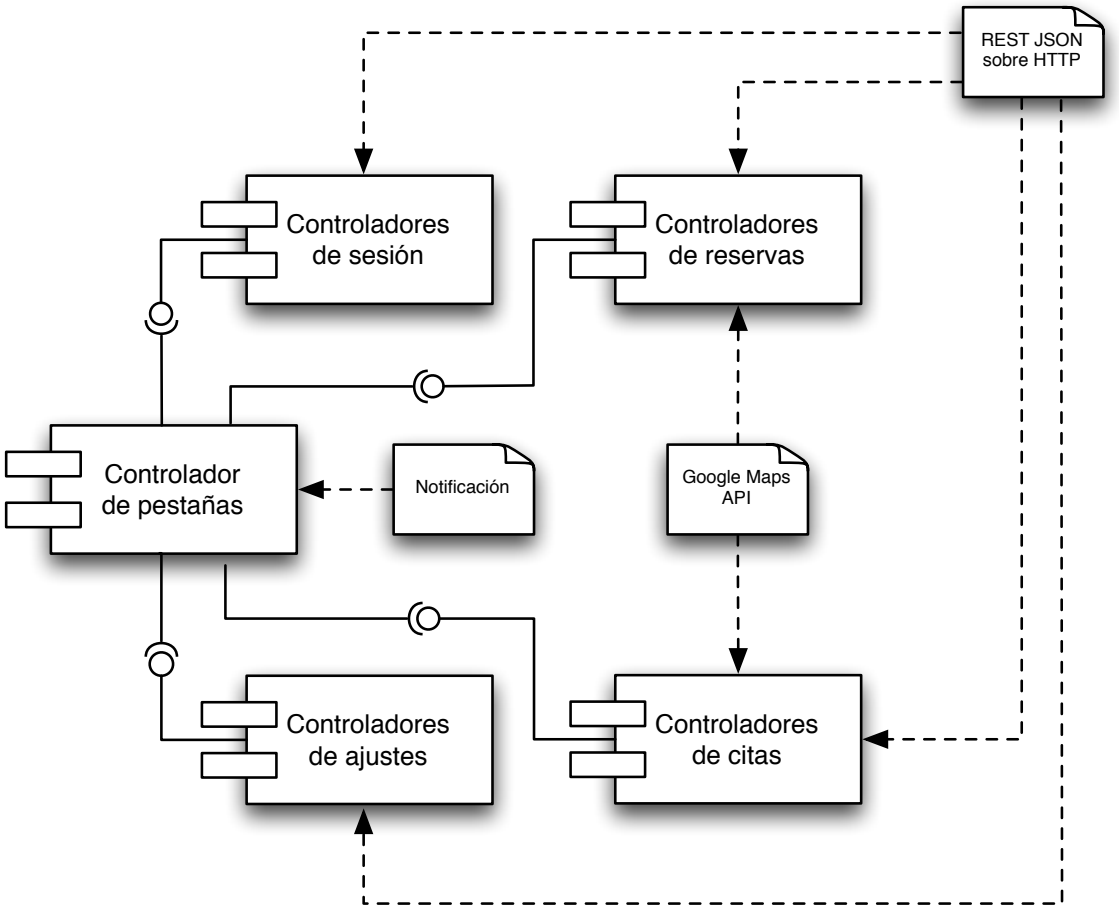

Figura 37. Componentes en la arquitectura de la capa de aplicación.

Notar en la Figura 37 que no está representado un modelo. Esto es porque el acceso de los controladores de esta capa a la base de datos se realiza siempre a través de la API móvil mediante peticiones HTTP y arquitectura REST con JSON como formato de representación[28].

[28] Ver apartado 3.1

Se han definido cuatro grupos de controladores dentro de la aplicación móvil de servicio con el fin de proporcionar los suficientes métodos para ejecutar los casos de uso (ver Figura 2) de un usuario final:

- Controladores de sesión.
- Controladores de citas.
- Controladores de reservas.
- Controladores de ajustes.

Todos ellos son manejados a su vez por un controlador de pestañas propio del sistema operativo iOS[29], cuya función es mostrar en pantalla la vista del controlador que el usuario final haya requerido mediante un toque con su dedo en una de las pestañas que aparecen en la parte inferior de la aplicación.

3.2.1. Controladores de sesión

Este grupo comprende a aquellos controladores (ver Figura 38) encargados de gestionar el inicio de sesión de un usuario final, el registro de nuevas cuentas y el restablecimiento de la contraseña, todo ello desde una interfaz móvil con comunicación al modelo de datos del servidor.

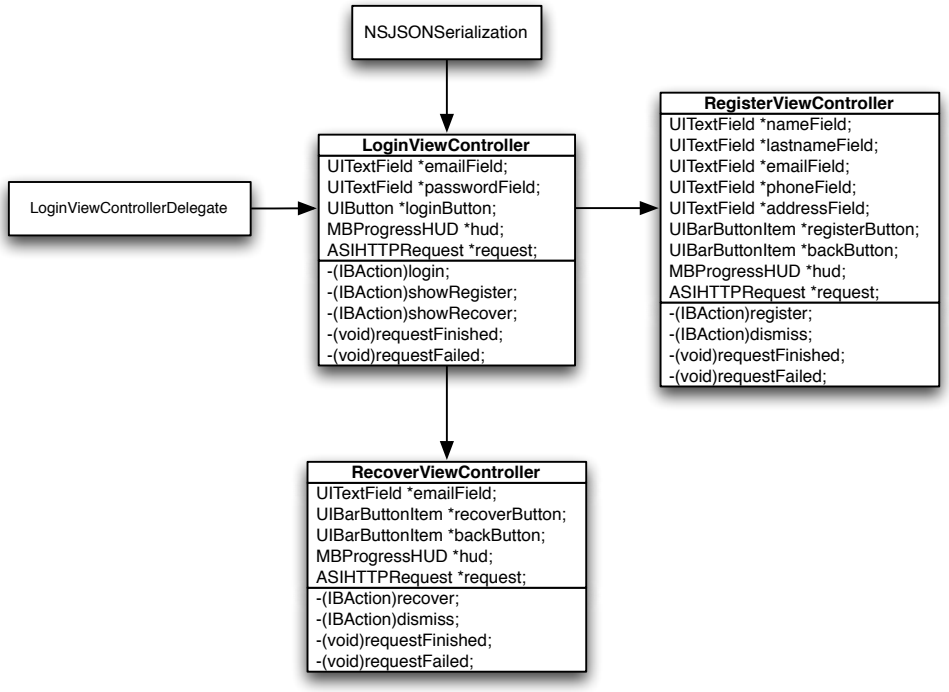

Figura 38. Diagrama de clases: controladores de sesión.

[29] Más información en [20]

La descripción de cada controlador y de sus métodos y atributos es la siguiente:

LoginViewController	
Gestiona el inicio de sesión.	
Atributos	
UITextField *emailField	Campo de texto donde introducir el identificador único de usuario final (correo electrónico).
UITextField *passwordField	Campo de texto donde introducir la contraseña personal.
UIButton *loginButton	Botón para comenzar una sesión.
UIBarButtonItem *registerButton	Botón en la barra de navegación para crear una nueva cuenta.
UIBarButtonItem *recoverButton	Botón en la barra de navegación para restaurar una contraseña olvidada.
MBProgressHUD *hud	Indicador de progreso durante la conexión con el servidor.
ASIHTTPRequest *request	Petición REST sobre HTTP.
Métodos	
(IBAction)login	El usuario final pulsa el botón de comienzo de sesión, lanzando la petición al servidor con los datos introducidos en los campos de texto.
(IBAction)showRegister	El usuario final pulsa el botón de crear nueva cuenta. Se inicia una instancia del *RegisterViewController* y se muestra su interfaz en pantalla.
(IBAction)showRecover	El usuario final pulsa el botón de restaurar su contraseña. Se inicia una instancia del *RecoverViewController* y se muestra su interfaz en pantalla.

(void)requestFinished	La petición de inicio de sesión es devuelta correctamente por el servidor con *statusCode = 200.* Se parsean los datos personales que llegan en la respuesta de la petición con la clase NSJSONSerialization y se almacenan en un registro interno de seguridad existente en iOS (KeyChain). La vista de este controlador desaparece y se genera un nuevo controlador de pestañas que gestiona el resto de los grupos de controladores. A continuación aparece la vista con la lista de citas reservadas por el usuario final que ha iniciado correctamente la sesión.
(void)requestFailed	La petición de inicio de sesión devuelve un error.

Tabla 32. Controladores de sesión (iPhone): LoginViewController.

RegisterViewController	
Permite la creación de una nueva cuenta de usuario final.	
Atributos	
UITextField *nameField	Campo de texto donde introducir el nombre.
UITextField *lastnameField	Campo de texto donde introducir los apellidos.
UITextField *emailField	Campo de texto donde introducir el identificador único de usuario final (correo electrónico).
UITextField *phoneField	Campo de texto donde introducir el teléfono.
UItextField *addressField	Campo de texto donde introducir la dirección.
UIBarButtonItem *registerButton	Botón en la barra de navegación para confirmar la creación de una nueva cuenta.
UIBarButtonItem *backButton	Botón en la barra de navegación para volver a la vista de inicio de sesión (*LoginViewController*).
MBProgressHUD *hud	Indicador de progreso durante la conexión con el servidor.
ASIHTTPRequest *request	Petición REST sobre HTTP.
Métodos	
(IBAction)register	El usuario final pulsa el botón de confirmar la creación de una nueva cuenta. Se lanza la petición al servidor sobre HTTP incluyendo los datos introducidos en los campos de texto.

(IBAction)dismiss	El usuario final pulsa el botón de volver a la vista de inicio de sesión. La vista de este controlador desaparece de la pantalla.
(void)requestFinished	La petición de creación de una cuenta de usuario final es devuelta correctamente por el servidor con *statusCode = 200.* Un e-mail de confirmación es enviado automáticamente a la cuenta de correo introducida en el campo correspondiente.
(void)requestFailed	La petición de creación de una cuenta de usuario final devuelve un error.

Tabla 33. Controladores de sesión (iPhone): RegisterViewController.

RecoverViewController	
Permite al usuario final pedir la recuperación de su contraseña.	
Atributos	
UITextField *emailField	Campo de texto donde introducir el identificador único de usuario final (correo electrónico).
UIBarButtonItem *recoverButton	Botón en la barra de navegación para solicitar la recuperación de la contraseña.
UIBarButtonItem *backButton	Botón en la barra de navegación para volver a la vista de inicio de sesión (*LoginViewController*).
MBProgressHUD *hud	Indicador de progreso durante la conexión con el servidor.
ASIHTTPRequest *request	Petición REST sobre HTTP.
Métodos	
(IBAction)recover	El usuario final pulsa el botón de recuperación de contraseña. Se lanza la petición al servidor sobre HTTP incluyendo el identificador introducir en el campo de texto correspondiente.
(IBAction)dismiss	El usuario final pulsa el botón de volver a la vista de inicio de sesión. La vista de este controlador desaparece de la pantalla.

(void)requestFinished	La petición de recuperación de contraseña es devuelta correctamente por el servidor con *statusCode = 200.* Un e-mail con un enlace a una página web donde establecer una nueva contraseña es enviado automáticamente a la cuenta de correo introducida en el campo correspondiente.
(void)requestFailed	La petición de recuperación de contraseña devuelve un error.

Tabla 34. Controladores de sesión (iPhone): RecoverViewController.

3.2.2. Controladores de citas

Este grupo gestionará todas las operaciones relacionadas con las citas ya reservadas por el usuario final. Incluye los controladores necesarios para imprimir un recibo de cita o visualizar la ruta en mapa desde la posición actual hasta el lugar de una cita.

En la Figura 39, se muestra el diagrama de clases de todos los controladores perteneciente a este grupo.

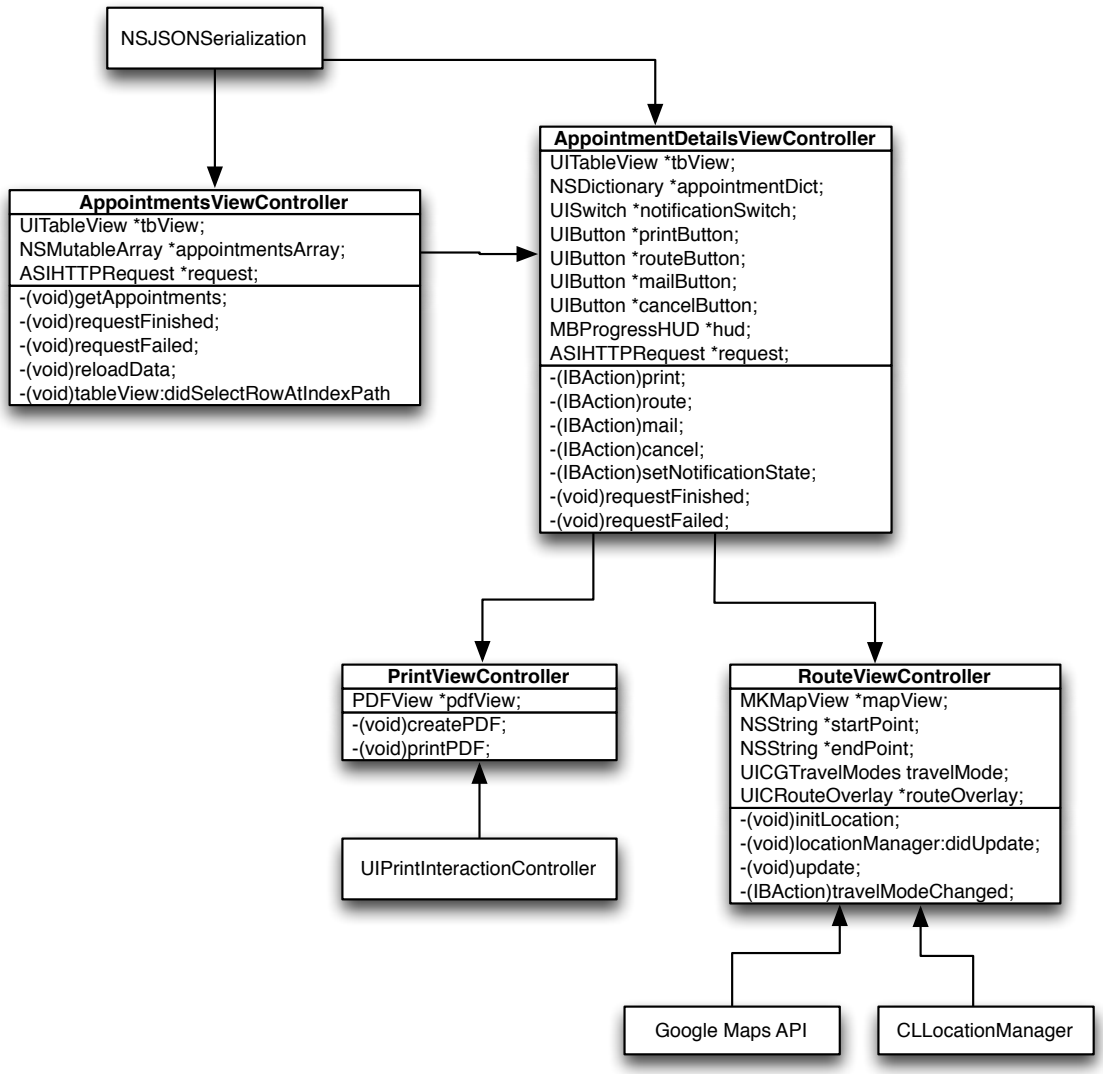

Figura 39. Diagrama de clases: controladores de citas.

El detalle de cada clase queda de la siguiente manera:

AppointmentsViewController	
Permite al usuario final visualizar el listado de todas sus citas reservadas.	
Atributos	
UITableView *tbView	Vista de tabla donde representar las citas reservadas del usuario final.
NSMutableArray *appointmentsArray	Array que almacena las citas reservas del usuario final una vez conseguidas del servidor. Cada objeto de este array (cita) es representado en una celda de la tabla.
ASIHTTPRequest *request	Petición REST sobre HTTP.

Métodos	
(void)getAppointments	Cada vez que este controlador es iniciado, se lanza este método cuya finalidad es lanzar la petición REST sobre HTTP al servidor para la recepción de las citas ya reservadas por el usuario final.
(void)requestFinished	La petición REST lanzada en el método anterior es devuelta correctamente por el servidor con *statusCode = 200* y con el JSON de las citas reservadas por el usuario. Este último se parsea con la clase *NSJSONSerialization* para generar el array del atributo *appointmentsArray*. Por último, se ejecuta el método *reloadData*.
(void)requestFailed	La petición REST lanzada en el primer método devuelve un error.
(void)reloadData	Refresca el contenido de la tabla de modo que el contenido de *appointmentsArray* sea visualizado correctamente en ella.
(void)tableView:didSelectRowAtIndexPath	El usuario final pulsa en una celda de la tabla para ver el detalle de una cita en concreto. Se crea una instancia del *AppointmentDetailsViewController* y su interfaz pasa al primer plano de la aplicación.

Tabla 35. Controladores de citas (iPhone): AppointmentsViewController.

AppointmentDetailsViewController	
Permite visualizar la información detallada de una cita reservada por el usuario final y las acciones correspondientes a ella.	
Atributos	
UITableView *tbView	Vista de tabla donde representar los datos de la cita reservada.
NSDictionary *appointmentDict	Diccionario (contenedor que almacena información mediante una asociación clave-valor) de la cita reservada.
UISwitch *notificationSwitch	Interruptor para activar o desactivar la recepción de notificaciones sobre el estado de la cola a la que pertenece la cita reservada.

UIButton *printButton	Botón para imprimir un recibo de la cita.
UIButton *routeButton	Botón para mostrar la ruta hasta el lugar de la cita.
UIButton *mailButton	Botón para recibir un e-mail con la información de la cita.
UIButton *cancelButton	Botón para cancelar la cita.
MBProgressHUD *hud	Indicador de progreso durante la conexión con el servidor.
ASIHTTPRequest *request	Petición REST sobre HTTP.
Métodos	
(IBAction)print	El usuario final pulsa el botón “imprimir recibo”. Se inicia una nueva instancia del *PrintViewController* y su interfaz aparece en pantalla.
(IBAction)route	El usuario final pulsa el botón “ver ruta”. Se inicia una nueva instancia del *RouteViewController* y su interfaz aparece en pantalla.
(IBAction)mail	Se lanza una petición REST sobre HTTP para que el servidor envíe un e-mail con los datos de la cita al correo electrónico registrado del usuario final.
(IBAction)cancel	Se lanza una petición REST sobre HTTP para cancelar la reserva de la cita en el sistema y ésta pase nuevamente a estar disponible.
(IBAction)setNotificationState	Cuando el usuario final cambia el valor del interruptor de notificaciones (*on* y *off)* se lanza una petición REST sobre HTTP para reflejar el cambio en el sistema.
(void)requestFinished	Todas las peticiones REST son devueltas correctamente con el *statusCode = 200.* Los mensajes de confirmación son parseados con la clase NSJSONSerialization y su contenido son mostrados en pantalla mediante una vista de alerta.
(void)requestFailed	Alguna de las peticiones REST lanzadas al servidor devuelven un error.

Tabla 36. Controladores de citas (iPhone): AppointmentDetailsViewController.

RouteViewController	
Permite visualizar la ruta en un mapa desde el punto en el que se encuentra el usuario final hasta el sitio donde tiene lugar una cita reservada por éste.	
Atributos	
MKMapView *mapView	Vista del mapa donde se representa la ruta.
NSString *startPoint	Cadena que almacena la posición inicial de la ruta, que es la posición actual del dispositivo móvil.
NSString *endPoint	Cadena que almacena la posición final de la ruta, que es el lugar donde se ejecuta la cita reservada.
UICGTravelModes travelMode	Representa el modo de viaje. Toma dos valores: *UICGTravelModelDriving* (en coche) y *UICGTravelModelWalking* (a pie)
UISegmentedControl *segControl	Selector en pantalla del modo de viaje: en coche o a pie.
Métodos	
(void)initLocation	Una vez iniciada la instancia de este controlador, se comienza con la localización del dispositivo mediante una nueva instancia del *CLLocationManager* (propio del SDK de iOS) encargado de gestionar el hardware GPS del dispositivo móvil.
(void)locationManager: didUpdate	La instancia del *CLLocationManager* devuelve la localización exacta del dispositivo y almacena ésta en el atributo *startPoint*. Lanza el método *update.*
update	Descarga de la API de Google Maps la ruta correspondiente desde el *startPoint* hasta el *endPoint* (el valor de éste atributo proviene del *AppointmentDetailsViewController*) con el modo de viaje preseleccionado. Traza la línea de la ruta aconsejada en el mapa y escribe el tiempo estimado de llegada en la barra de navegación.
(IBAction)travelMode Changed	Cuando el usuario final cambia el valor del selector de modo de viaje, el último valor seleccionado se almacena en el atributo *travelMode*. Lanza el método *initLocation*.

Tabla 37. Controladores de citas (iPhone): RouteViewController.

PrintViewController	
Permite imprimir el recibo de una cita reserva por el usuario final desde su dispositivo móvil sin necesidad de cables.	
Atributos	
PDFView *pdfView	Vista en PDF del recibo de la cita reservada.
Métodos	
(void)createPDF	Una vez iniciada la instancia de este controlador, se genera la vista PDF (*pdfView*) del recibo de la cita y se muestra en pantalla.
(void)printPDF	Se inicia el *UIPrintInteractionController* (propio del SDK de iOS) con el que seleccionar una impresora disponible y ejecutar la impresión del contenido en *pdfView*.

Tabla 38. Controladores de citas (iPhone): PrintViewController.

3.2.3. Controladores de reservas

El objetivo de este grupo de controladores es el permitir la reserva y confirmación por parte de un usuario final de una cita en cualquiera de las actividades activas. En la Figura 40 aparecen los controladores diseñados para esa tarea.

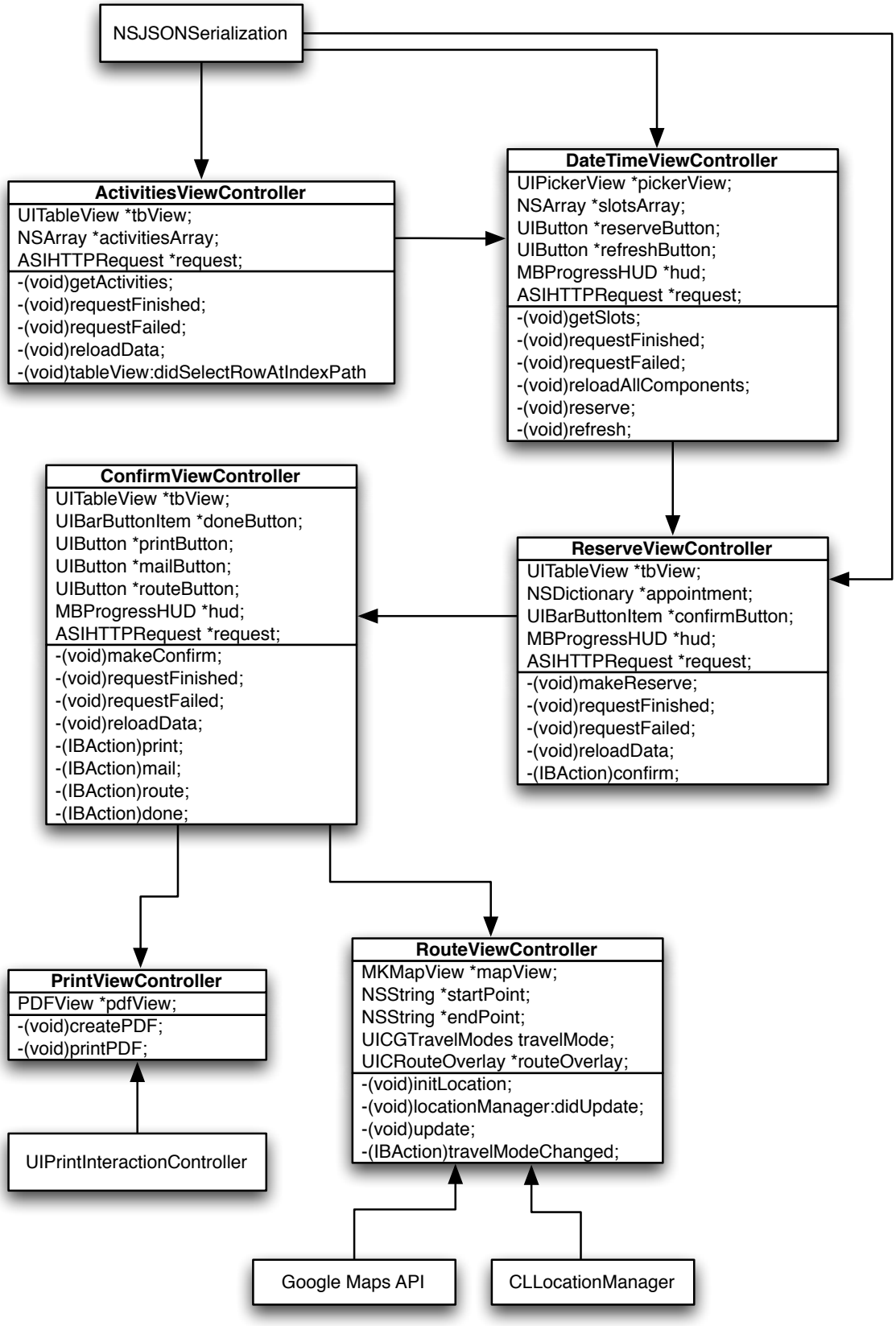

Figura 40. Diagrama de clases: controladores de reservas.

Los métodos y atributos de cada clase son los siguientes:

ActivitiesViewController	
Permite al usuario final visualizar el listado de todas las actividades activas para una reserva de cita.	
Atributos	
UITableView *tbView	Vista de tabla donde representar las actividades activas del sistema.
NSArray *activitiesArray	Array que almacena las actividades activas una vez conseguidas del servidor. Cada objeto de este array (actividad) es representado en una celda de la tabla.
ASIHTTPRequest *request	Petición REST sobre HTTP.
Métodos	
(void)getActivities	Cada vez que este controlador es iniciado, se ejecuta este método cuya finalidad es lanzar la petición REST sobre HTTP al servidor para la recepción de las actividades activas dentro del sistema.
(void)requestFinished	La petición REST lanzada en el método anterior es devuelta correctamente por el servidor con *statusCode = 200* y con el JSON de las actividades activas. Este último se parsea con la clase *NSJSONSerialization* para generar el array del atributo *activitiesArray*. Concluye llamando al método *reloadData*.
(void)requestFailed	La petición REST lanzada en el primer método devuelve un error.
(void)reloadData	Refresca el contenido de la tabla de modo que el contenido de *activitiesArray* sea visualizado correctamente en ella.
(void)tableView:didSelectRowAtIndexPath	El usuario final pulsa en una celda de la tabla para comenzar el proceso de reserva de una cita disponible para la actividad seleccionada. Se crea una instancia del *DateTimeViewController* y su interfaz pasa al primer plano de la aplicación.

Tabla 39. Controladores de reservas (iPhone): ActivitiesViewController.

DateTimeViewController	
Permite al usuario final seleccionar una cita disponible para su reserva.	
Atributos	
UIPickerView *pickerView	Vista de selector de horarios con citas disponibles.
NSArray *slotsArray	Array que contiene las citas disponibles para ser reservadas por un usuario final dentro de una actividad. Cada objeto de este array (cita) es representado en una fila del *pickerView* por su horario.
UIButton *reserveButton	Botón para avanzar al siguiente paso de reserva.
UIButton *refreshButton	Botón para refrescar las citas disponibles
MBProgressHUD *hud	Indicador de progreso durante la conexión con el servidor.
ASIHTTPRequest *request	Petición REST sobre HTTP.
Métodos	
(void)getSlots	Lanza la petición REST sobre HTTP al servidor para la recepción de las citas disponibles en la actividad seleccionada en el *ActivitiesController*.
(void)requestFinished	La petición REST lanzada en el método anterior es devuelta correctamente por el servidor con *statusCode = 200* y con el JSON de las citas disponibles. Este último se parsea con la clase *NSJSONSerialization* para generar el array del atributo *slotsArray*. Concluye llamando al método *reloadAllComponents*.
(void)requestFailed	La petición REST lanzada en el primer método devuelve un error.
(void)reloadAllComponents	Refresca el contenido del pickerView con el contenido de *slotsArray*.
(IBAction)reserve	Cuando el usuario final pulsa el botón de "*Reservar*" se crea una instancia del *ReserveViewController* y su interfaz pasa al primer plano en la pantalla.

(IBAction)refresh	Cuando el usuario final pulsa el botón de "*Refrescar*" se llama al método *getSlots.*

Tabla 40. Controladores de reservas (iPhone): DateTimeViewController.

ReserveViewController	
Realiza la reserva de una cita disponible cuyo horario ha sido seleccionado en el controlador *DateTimeViewController*.	
Atributos	
UITableView *tbView	Vista de tabla donde representar los datos de la cita reservada.
NSDictionary *appointment	Diccionario que contiene la información de la cita reservada.
UIBarButtonItem *confirmButton	Botón en la barra de navegación para avanzar al siguiente paso de confirmación.
MBProgressHUD *hud	Indicador de progreso durante la conexión con el servidor.
ASIHTTPRequest *request	Petición REST sobre HTTP.
Métodos	
(void)makeReserve	Cada vez que el controlador es iniciado, se ejecuta este método cuya finalidad es lanzar la petición REST sobre HTTP al servidor para realizar la reserva de la cita en el sistema.
(void)requestFinished	La petición REST lanzada en el método anterior es devuelta correctamente por el servidor con *statusCode = 200* y con el JSON de la cita ya reservada. Este último se parsea con la clase *NSJSONSerialization* para generar el diccionario del atributo *appointment*. Concluye llamando al método *reloadData*.
(void)requestFailed	La petición REST lanzada en el primer método devuelve un error.
(void)reloadData	Refresca el contenido de la tabla de modo que el contenido de *appointment* sea visualizado correctamente en ella.
(IBAction)confirm	Cuando el usuario final pulsa el botón de "*Confirmar*" se crea una instancia del *ConfirmViewController* y su interfaz pasa al primer plano en la pantalla.

Tabla 41. Controladores de reservas (iPhone): ReserveViewController.

ConfirmViewController	
Confirma la reserva de la cita del controlador *ReserveViewController*.	
Atributos	
UITableView *tbView	Vista de tabla donde representar los datos de la cita confirmada.
UIBarButtonItem *doneButton	Botón en la barra de navegación para dar por terminado el proceso de reserva.
UIButton *printButton	Botón para imprimir el recibo de la cita confirmada.
UIButton *mailButton	Botón para solicitar el envío de un e-mail con los datos de la cita confirmada a la dirección de correo del usuario final.
UIButton *routeButton	Botón para mostrar la ruta hasta el lugar de la cita confirmada.
MBProgressHUD *hud	Indicador de progreso durante la conexión con el servidor.
ASIHTTPRequest *request	Petición REST sobre HTTP.
Métodos	
(void)makeConfirm	Una vez el controlador es iniciado, se ejecuta este método cuya finalidad es lanzar la petición REST sobre HTTP al servidor para realizar la confirmación de la reserva realizada en *ReserveViewController*.
(void)requestFinished	La petición REST lanzada en el método anterior es devuelta correctamente por el servidor con *statusCode = 200.* Concluye llamando al método *reloadData*.
(void)requestFailed	La petición REST lanzada en el primer método devuelve un error.
(void)reloadData	Refresca el contenido de la tabla con el contenido de *appointment* perteneciente al *ReserveViewController.* Además, se añade una celda más con un icono indicador de que la confirmación se ha realizado con éxito.
(IBAction)print	Cuando el usuario final pulsa el botón de "*Imprimir recibo*" se crea una instancia del *PrintViewController* y su interfaz pasa al primer plano de la pantalla.

(IBAction)mail	Cuando el usuario pulsa el botón de "Enviar e-mail" se solicita al servidor mediante una petición REST el envío de un e-mail con los datos de la cita confirmada. No se espera respuesta alguna de esta petición.
(IBAction)route	Cuando el usuario final pulsa el botón de "*Ver Ruta*" se crea una instancia del *RouteViewController* y su interfaz pasa al primer plano de la pantalla.
(IBAction)done	Cuando el usuario final pulsa el botón de "Terminado" se vuelve al controlador *ActivitiesController,* apareciendo nuevamente su interfaz en la pantalla.

Tabla 42. Controladores de reservas (iPhone): ConfirmViewController.

3.2.4. Controladores de ajustes

Este grupo de controladores sirve para permitir al usuario modificar sus datos personales en la base de datos del servidor directamente desde la aplicación móvil de servicio. Además incluye las acciones de cerrar la sesión activa del usuario final y la de borrar una cuenta del sistema.

En la Figura 41, se muestra con detalle los controladores que conforman este grupo.

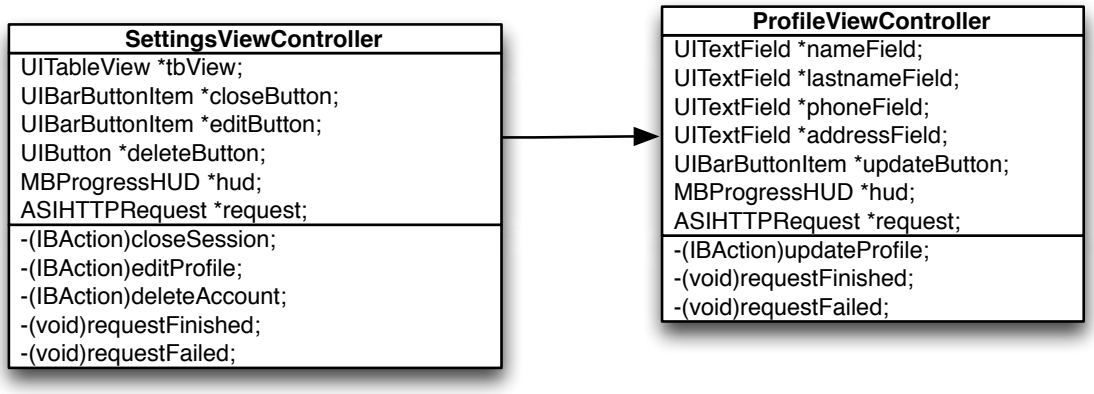

Figura 41. Diagrama de clases: controladores de ajustes.

Los métodos y atributos más destacables en cada controlador son los siguientes:

SettingsViewController	
Permite al usuario final cerrar su sesión privada en la aplicación móvil y borrar su cuenta del sistema.	
Atributos	
UITableView *tbView	Vista de tabla donde representar los datos de la sesión abierta. Contiene dos celdas: una para mostrar el identificador único de sesión (correo electrónico) y otra para mostrar los nombres y apellidos del usuario final.
UIBarButtonItem *closeButton	Botón en la barra de navegación para cerrar la sesión privada de usuario final.
UIBarButtonItem *editButton	Botón en la barra de navegación para editar el perfil en el sistema del usuario final con sesión abierta en la aplicación móvil de servicio.
UIButton *deleteButton	Botón para borrar la cuenta del usuario final.
MBProgressHUD *hud	Indicador de progreso durante la conexión con el servidor.
ASIHTTPRequest *request	Petición REST sobre HTTP.
Métodos	
(IBAction)closeSession	Cuando el usuario pulsa el botón de cerrar sesión se borran todos los datos temporales y se carga el *LoginViewController* con su interfaz en pantalla.
(IBAction)editProfile	Cuando el usuario pulsa el botón de editar perfil se inicia una instancia del *ProfileViewController* y su interfaz aparece en pantalla.
(IBAction)deleteAccount	Cuando el usuario pulsa el botón de borrar cuenta del sistema, se lanza una petición REST sobre HTTP para que el servidor efectúe las operaciones necesarias.
(void)requestFinished	La petición HTTP del método anterior es devuelta correctamente con el *statusCode = 200*. Se lanza el método *closeSession*.
(void)requestFailed	La petición REST lanzada al servidor devuelven un error.

Tabla 43. Controladores de ajustes (iPhone): SettingsViewController.

ProfileViewController	
Permite a un usuario final modificar sus datos personales del perfil almacenado en la base de datos del sistema.	
Atributos	
UITextField *nameField	Campo de texto donde introducir el nombre.
UITextField *lastnameField	Campo de texto donde introducir los apellidos.
UITextField *phoneField	Campo de texto donde introducir el teléfono.
UITextField *addressField	Campo de texto donde introducir la dirección.
UIBarButtonItem *updateButton	Botón en la barra de navegación para lanzar la actualización en el sistema del perfil del usuario final.
MBProgressHUD *hud	Indicador de progreso durante la conexión con el servidor.
ASIHTTPRequest *request	Petición REST sobre HTTP.
Métodos	
(IBAction)updateProfile	Cuando el usuario pulsa el botón de actualizar perfil se lanza una petición REST sobre HTTP al servidor con los valores presentes en cada uno de los campos de texto.
(void)requestFinished	La petición HTTP del método anterior es devuelta correctamente con el *statusCode = 200*. Se muestra un mensaje de confirmación, y se almacenan en un registro interno del teléfono los nuevos datos personales del usuario final.
(void)requestFailed	la petición REST lanzada al servidor devuelve un error.

Tabla 44. Controladores de ajustes (iPhone): ProfileViewController.

3.3. Diagramas de secuencia

En los siguientes apartados se tratará de mostrar mediante diagramas de secuencia la interacción entre un usuario final y el sistema para entender aún mejor el diseño y la arquitectura de este último. Con el fin de no alargar en exceso esta memoria, sólo serán representados aquellos escenarios más habituales para el usuario final.

3.3.1. Inicio de sesión de un usuario final

En la Figura 42 se muestra el diagrama de secuencia para el inicio de sesión en la aplicación móvil por parte de un usuario final. En su mayoría, los objetos pertenecen a la arquitectura móvil a excepción del objeto Mobile::LoginController[30] que pertenece a la capa de servidor, en particular al grupo de controladores para la API móvil, y cuya función será la de comprobar las credenciales enviadas desde el dispositivo en la base de datos permitiendo posteriormente el acceso a la aplicación para la visualización y reserva de citas.

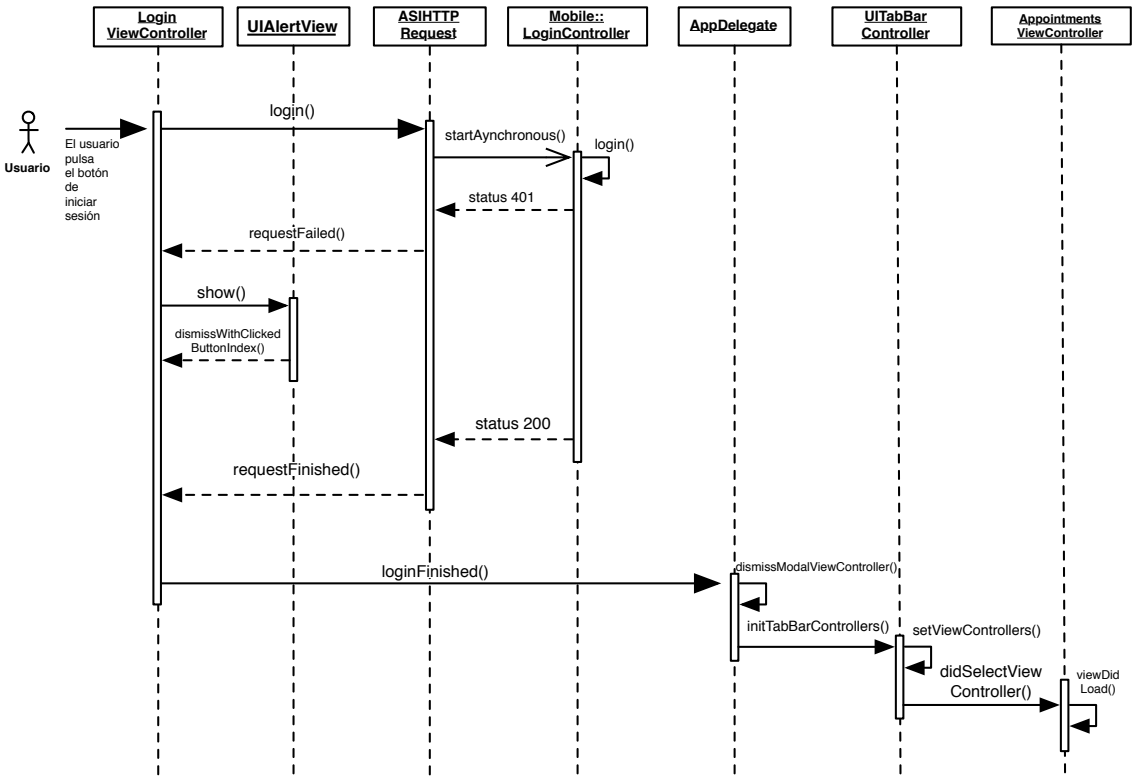

Figura 42: Diagrama de secuencia: inicio de sesión de un usuario final.

Resaltar el objeto UIAlertView por pertenecer al SDK de iOS, y cuya misión es la de mostrar una alerta en pantalla con un texto descriptivo y uno o varios botones de acción.

3.3.2. Reserva de una cita

En la Figura 43, se muestra la interacción entre componentes de las dos capas del sistema cuando un usuario final reserva una cita de su dispositivo móvil. Como se puede ver, el usuario final solo necesita de cinco acciones:

[30] La nomenclatura Mobile::LoginController en Ruby on Rails indica que el controlador con clase *LoginController* se encuentra dentro de la carpeta *Mobile.*

★ Elegir una actividad.
★ Seleccionar una cita disponible para su reserva.
★ Pulsa el botón "*Reservar*".
★ Pulsa el botón "*Confirmar"*.
★ Pulsa el botón "*Terminar".*

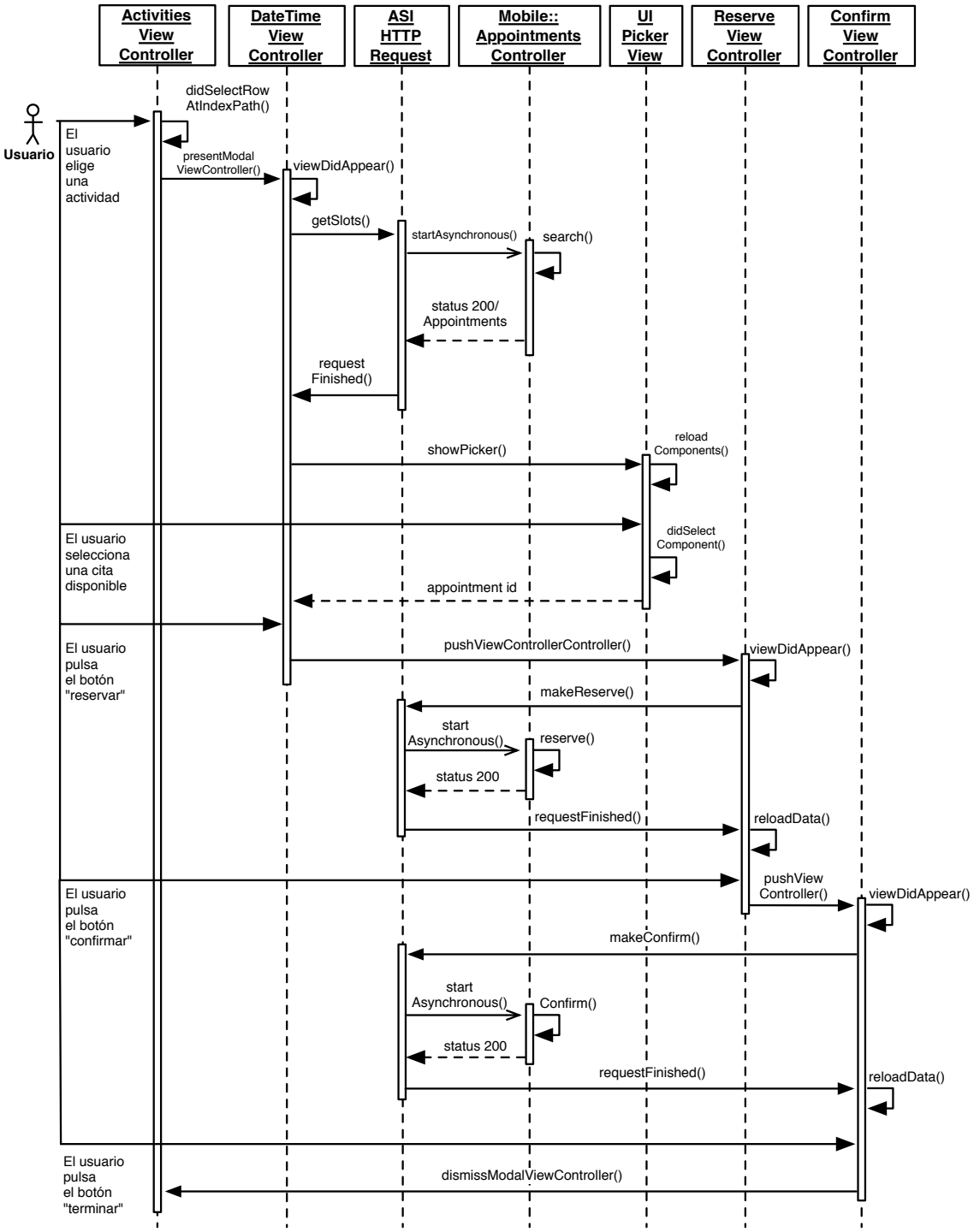

Figura 43. Diagrama de secuencia: reserva de una cita.

3.4. Diseño del ciclo de vida de una cita

La mayoría de las acciones realizadas en el sistema por los diferentes actores siempre involucran a una o varias citas. Por ejemplo: un usuario cancela una cita, un administrador crea una cola de citas cuando establece un turno de trabajo, un empleado comienza una cita, etc…

Para que el sistema sea capaz de identificar las acciones disponibles para cada cita se ha diseño una estructura de estados que define el ciclo de vida de cada una de ellas. De esta manera, una cita pasará por diferentes estados [31] cada vez que una acción determinada sea ejecutada, hasta que finalmente llegue a un estado final específico donde ya no podrá ser modificado.

En la Figura 44, se muestra un diagrama de estados que describe el ciclo de vida completo de una cita, desde que es creada por el administrador cuando éste genera una nueva cola de citas hasta el momento en que es finalizada por un empleado o bloqueada por un operador.

[31] Modificación del valor entero en el campo "*state*" para un registro de la tabla "*citas*".

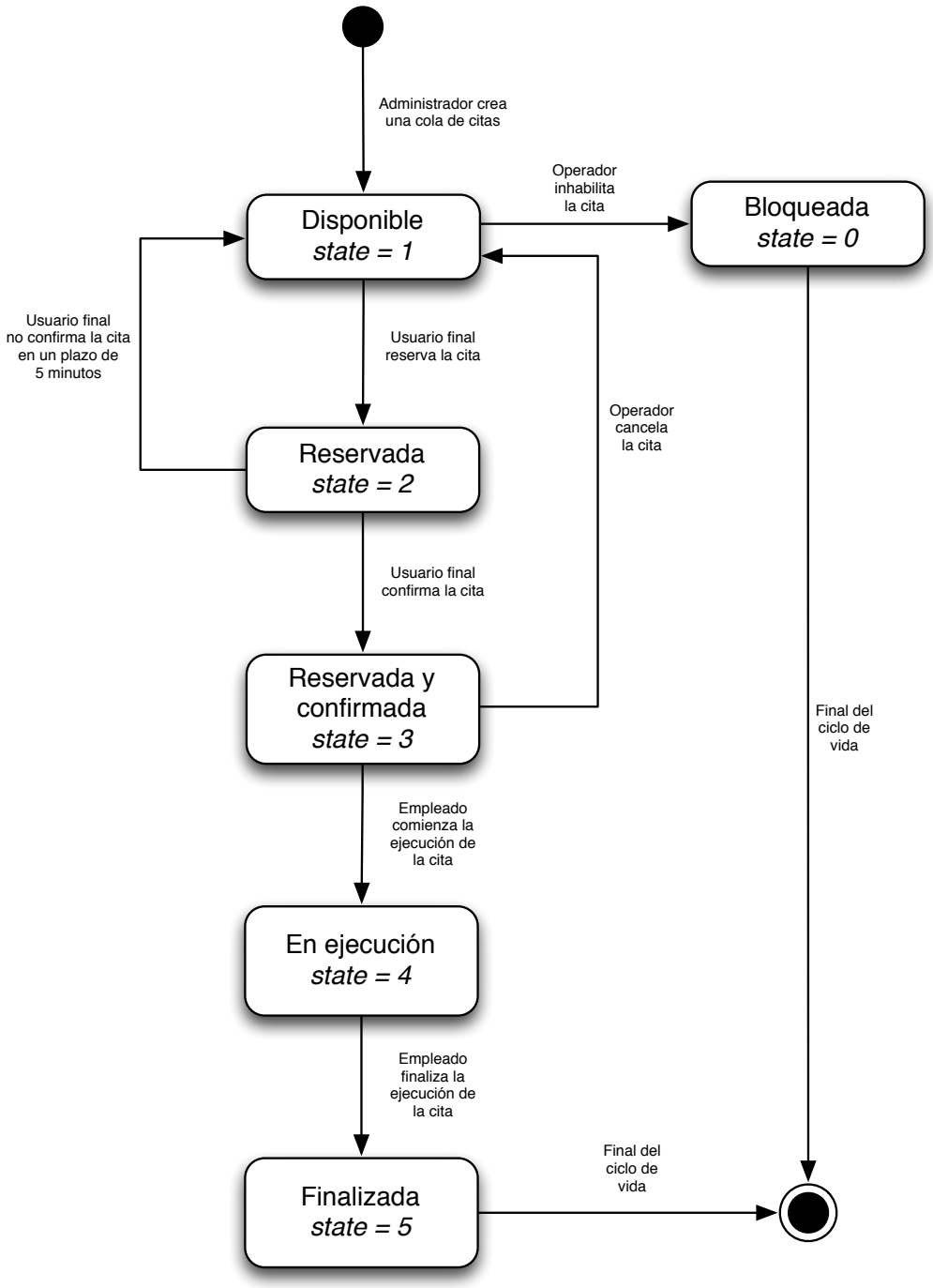

Figura 44. Diagrama de estados: ciclo de vida de una cita en el sistema.

3.5. Diseño del módulo de control automático para colas de citas

El requisito más importante que debe cumplir este proyecto, es el de incluir un módulo de control totalmente automatizado, cuya misión sea comprobar en tiempo real el estado de cada una de las colas de citas que se encuentren en ejecución por parte de un empleado, y a continuación, comunicarlo a los correspondientes usuarios con cita reservada, mediante una notificación directa a su dispositivo móvil. Una vez entendido

esto, se puede decir que el diseño del módulo de control automático para el estado de las colas de citas constará de dos operaciones básicas, que son:

1. **Cálculo del estado;** determina qué cantidad de adelanto o retraso en tiempo (estado) presenta una cola de citas.

2. **Notificación del estado;** determina qué usuarios deben de ser avisados sobre el estado calculado en la operación anterior, y se envía la correspondiente notificación estimando cuándo comenzarán realmente las citas.

En los siguientes apartados, se dará una visión más profunda del diseño de cada una de estas operaciones. Ésta será apoyada por una serie de diagramas de flujos, que han servido de base en la implementación del algoritmo escrito para este módulo.

3.5.1. Cálculo del estado

Durante la ejecución de una cola de citas por parte de un empleado, el parámetro que desea conocer cualquiera de los usuarios con una cita reservada en ella, es su estado.

Situándose en un escenario real, cuando un usuario llega al lugar de la actividad correspondiente a su cita, lo primero que quiere conocer es su adelanto o retraso. Es por esta razón, por la que la primera operación que debe de acometer el módulo de control es el cálculo del estado.

El diseño de esta operación parte de la base de que, para determinar el estado de una cola de citas en un instante de tiempo determinado, es necesario conocer la última acción realizada por el empleado, es decir, saber si éste se encuentra ejecutando una cita o no. Ello servirá para plantear dos escenarios diferentes (ver Figura 45) en los que el cálculo del estado dependa de una estimación sobre el tiempo de finalización de la cita en ejecución, o bien, de la siguiente cita dentro de la cola en posición de espera.

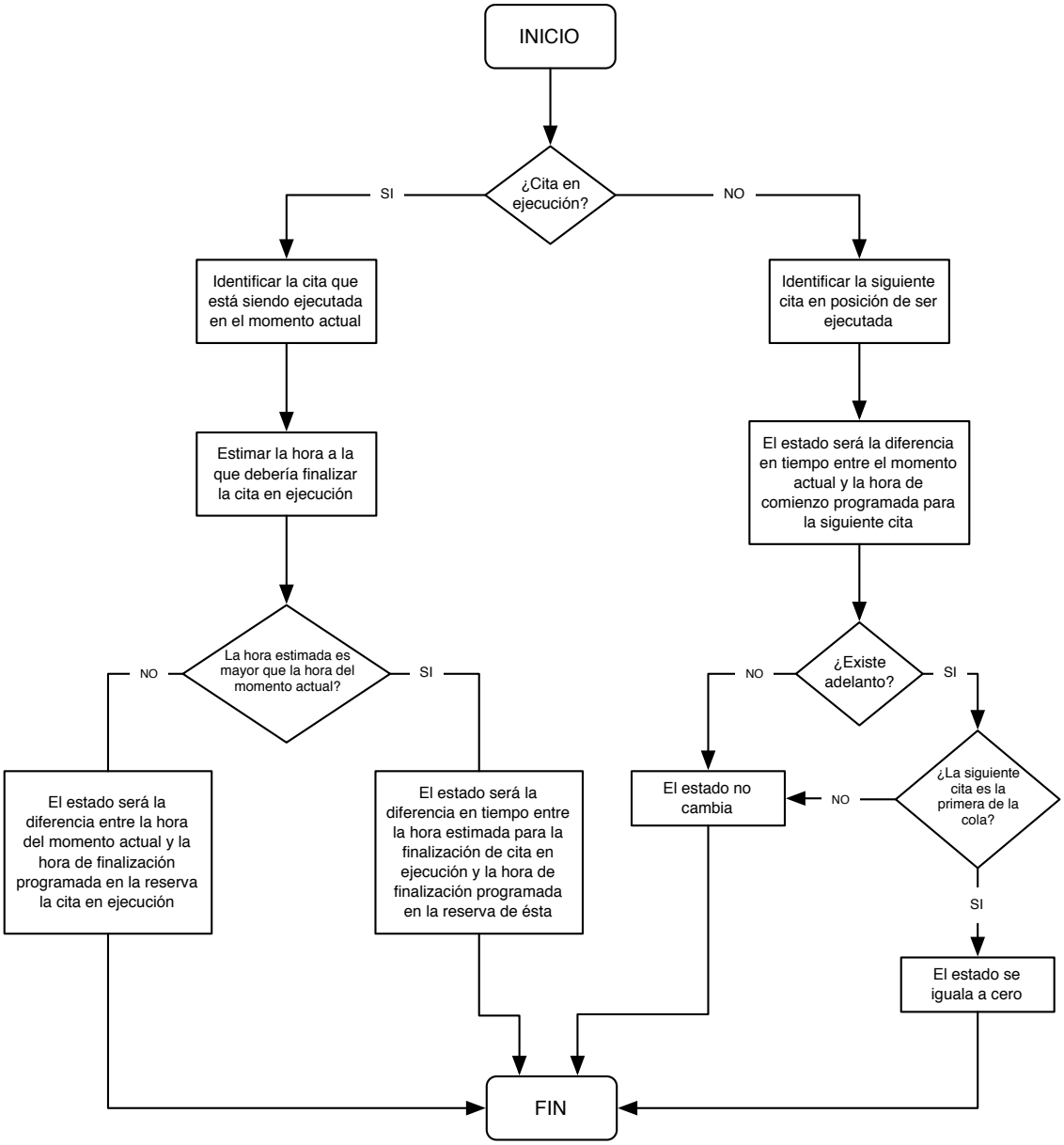

Figura 45. Diagrama de flujo: cálculo del estado de una cola de citas.

En la Figura 45, se muestra el diagrama de flujo que representa la secuencia de pasos a realizar por la operación de cálculo del estado en una cola de citas. Se pueden observar dos ramas verticales perfectamente delimitadas, las cuales, representan cada uno de los escenarios comentados en el párrafo anterior.

En la rama izquierda se contempla el caso en el que un empleado esté en medio de la ejecución de una cita cuando esta operación sea lanzada. La primera acción es obtener todos los datos correspondientes a dicha cita, para a continuación realizar una estimación de la hora en la que debería finalizar, mediante la adición del tiempo medio de cita establecido por el administrador a la hora en la que ésta fue comenzada por el empleado. Por ejemplo, si la cita en curso empezó a las 8:36 y el tiempo medio de cita para esa cola es de 5 minutos, se estima que su finalización ocurrirá a las 8:41. Cuando la estimación de finalizado de la cita supere al tiempo en el que se está realizando la

operación del cálculo del estado, este último será la diferencia en tiempo, ya sea minutos u horas, entre la estimación y el horario programado por defecto para la finalización de la cita. En la Tabla 45 y continuando con el ejemplo anterior, se muestra el resultado obtenido para el estado de la cola en el supuesto planteando hasta aquí.

Horario de cita programado	Hora de comienzo de cita	Hora estimada de finalización de cita	Hora de lanzamiento de la operación	Estado de cola (minutos)
8:20 - 8:25	8:36	8:41	8:40	16

Tabla 45. Ejemplo de una operación de cálculo de estado para el caso en que el tiempo estimado de finalización de la cita en ejecución supere al del lanzamiento del módulo de control automatizado para colas de citas.

Nótese que el estado de la cola sería la resta entre las 8:41 y las 8:25, esto es, un total de 16 minutos. Un valor positivo en el estado de cola significará que existe retraso, mientras que si es negativo implicará que hay adelanto. En caso de resultar cero, la cola de citas irá a tiempo con lo programado.

Cuando la hora estimada de finalización no supere a la del lanzamiento del módulo de control automatizado con su correspondiente operación de cálculo del estado, este último resultará de la diferencia en tiempo entre dicha hora de lanzamiento y la de finalización programada en el horario de la cita. Esto es así, porque el sistema siempre debe esperar que todas las citas finalicen dentro del tiempo programado para su ejecución (ver Tabla 45), y de no ser así, que lo hagan en un momento próximo al lanzamiento de la operación de cálculo de estado (ver Tabla 46).

Horario de cita programado	Hora de comienzo de cita	Hora estimada de finalización de cita	Hora de lanzamiento de la operación	Estado de cola (minutos)
8:20 - 8:25	8:36	8:41	8:45	20

Tabla 46. Ejemplo de una operación de cálculo de estado para el caso en que el tiempo estimado de finalización de la cita en ejecución sea inferior o igual al del lanzamiento del módulo de control automatizado para colas de citas.

Nótese que el estado de la cola sale de la resta entre las 8:45 y las 8:25, esto es, 20 minutos de retraso.

En el caso de que no existe una cita en ejecución por parte de un empleado (rama derecha en la Figura 45), lo primero a realizar será la obtención de los datos correspondientes a la siguiente cita que se encuentre en posición de espera dentro de la cola. De esta manera, el estado resultará de la diferencia entre el tiempo del instante en el que se está ejecutando el módulo de control automatizado y la hora de comienzo programada para dicha cita.

Horario programado de la siguiente cita en la cola	Hora de lanzamiento de la operación	Estado de cola (minutos)
8:20 - 8:25	8:30	10

Tabla 47. Ejemplo de una operación de cálculo de estado para el caso en que no se encuentre una cita en ejecución durante la actuación del módulo de control automatizado.

Si el estado de la cola resulta negativo y la siguiente cita en espera se corresponde con la primera a ejecutar dentro de la cola, éste automáticamente pasa a valer cero, evitando así el aviso de la existencia de un adelanto a los primeros usuarios finales. Si así fuese, éstos llegarían con demasiada antelación al comienzo de la cola por parte del empleado.

3.5.2. Notificación del estado

La siguiente operación básica a realizar tras el cálculo del estado comentado anteriormente, es la notificación del estado. Su objetivo es determinar qué citas deben ser avisadas y enviar dicha notificación a los dispositivos móviles asociados a éstas.

El diseño de esta operación surge de la idea de que solamente será necesario mandar notificaciones a aquellas citas más próximas en el tiempo, ya sea a continuación de una cita en ejecución o a partir de la siguiente cita en espera de ser comenzada por un empleado. Ante esta premisa, se establecerá una ventana de tiempo que recoja el grupo de citas susceptibles de ser notificadas, y cuyo tamaño dependerá del valor y signo del estado recibido desde la operación básica de cálculo anterior.

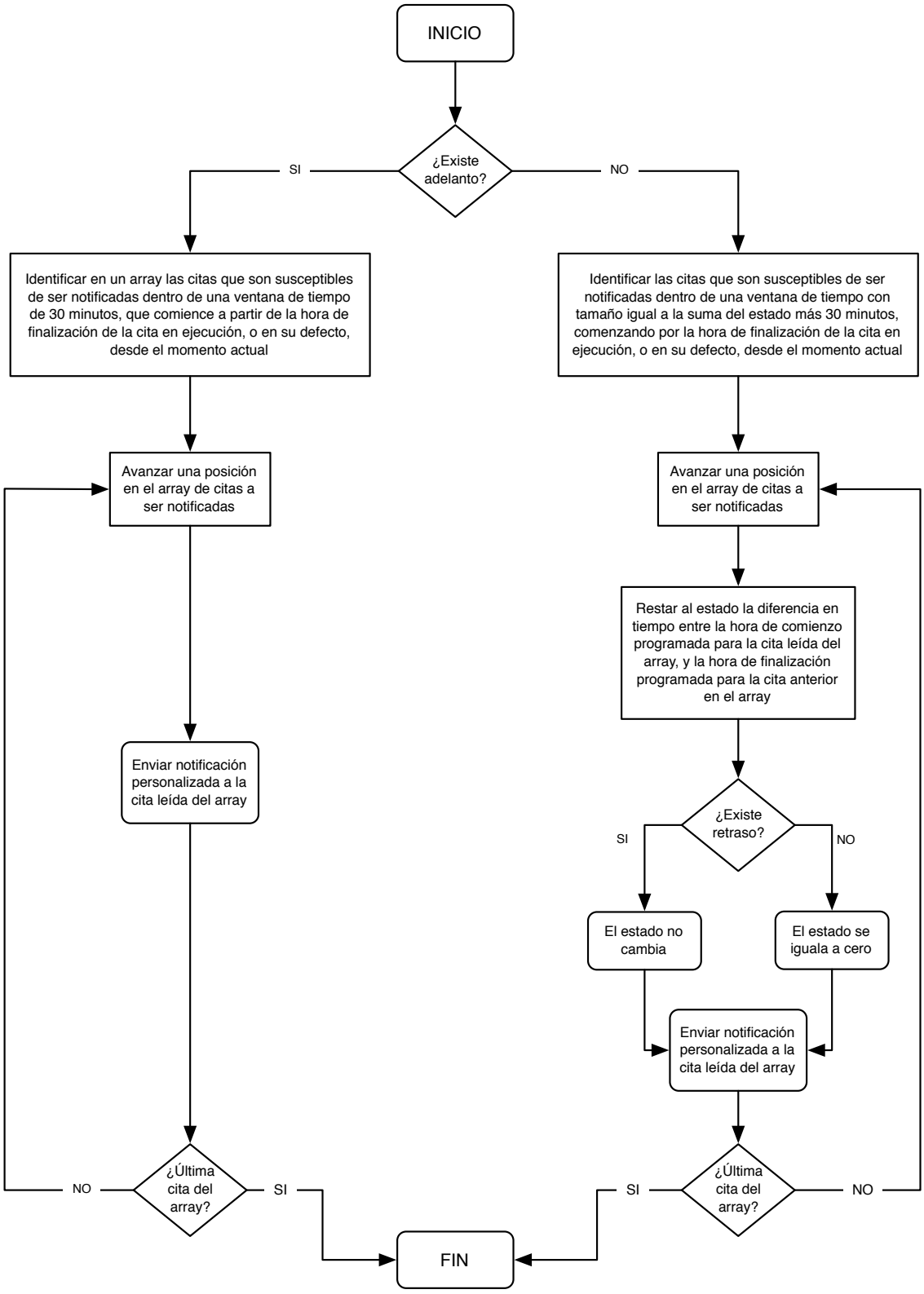

Figura 46. Diagrama de flujo: notificación del estado de una cola de citas.

Como se puede observar en la Figura 46, existe una bifurcación al comienzo de la operación destinada a la notificación del estado de una cola de citas. Su finalidad es separar el flujo de trabajo en dos ramas diferentes, según el signo del estado calculado.

En la rama izquierda de la figura, se describe los pasos a realizar para la notificación del estado cuando éste presente un signo negativo, es decir, haya un adelanto en la cola de citas. La primera acción será identificar y agrupar aquellas citas que se quieren avisar sobre un adelanto registrado. Para ello, se define una ventana de tiempo de tamaño igual a 30 minutos, cuyo origen se encuentra en la hora exacta del lanzamiento de la operación. A continuación, se filtran todas las citas reservadas en posición de espera con un horario de comienzo programado dentro de la ventana de tiempo y se almacenan en un *array*.

Una vez hayan sido identificadas las citas correspondientes para su aviso, se recorre el *array* fila por fila, generando un mensaje personalizado para cada cita con el valor de estado de la cola y la hora estimada de comienzo. En la Tabla 48, se muestra un ejemplo de una cola de citas con un adelanto de 12 minutos sobre la programación inicial. Los mensajes contendrían los valores correspondientes a las celdas de la última columna.

Hora de lanzamiento de la operación: 8:40			
Horario programado de cita en ejecución	**Hora de comienzo de cita en ejecución**	**Hora estimada de finalización de cita en ejecución**	**Estado de cola (minutos)**
8:50 - 8:55	8:38	8:43	-12
Horario programado de la siguientes citas en la cola y dentro de los límites de la ventana de tiempo			**Hora estimada de comienzo de cita**
8:55 - 9:00			8:43
9:05 - 9:10			8:53

Tabla 48. Ejemplo de una operación de notificación de estado para el caso en que la cola de citas presente adelanto respecto a lo programado.

Por otro lado, cuando el estado obtenido en la operación de cálculo no sea negativo, es decir, haya un retraso en la cola de citas o ésta vaya a tiempo, la operación de notificación seguirá el flujo de trabajo de la rama derecha que aparece en la Figura 46. Al igual que en el caso anterior, la primera acción será identificar qué citas van a ser avisadas del estado con la utilización de una ventana de tiempo, cuyo origen se encuentre en la hora programada para la finalización de la cita en ejecución, o en su defecto, sobre la hora del instante en el que sea lanzada esta operación. A diferencia de antes, el tamaño no será siempre constante sino que a los 30 minutos establecidos por defecto, habrá que añadirle el valor del estado para evitar que queden citas sin notificar. En la Tabla 49, se muestra un claro ejemplo de ello.

Horario programado de cita en ejecución	Hora de lanzamiento de la operación	Hora de comienzo de cita	Hora estimada de finalización de cita	Estado de cola (minutos)	Límites de la ventana de tiempo
8:10 - 8:15	8:40	8:37	8:42	27	8:15 - 9:12

Tabla 49. Ejemplo de límites de la ventana de tiempo para el caso en que la cola de citas presente retraso respecto a lo programado.

Notar que si no se añadiesen los 27 minutos de retraso en la cola de citas al tamaño de la ventana de tiempo, el límite superior de ésta quedaría en 8:45, lo cual, supondría que las citas programadas a partir de las 8:50 y en adelante no serían avisadas, siendo ya las 8:40. Observar además, que en el caso de que el estado valga cero, es decir, no haya ni retraso ni adelanto, la adición del estado al tamaño de la ventana resultaría indiferente, dejando a este último en los 30 minutos por defecto.

Una vez identificadas y agrupadas las citas susceptibles de ser notificadas, se iterará sobre ellas para crear un mensaje personalizado para cada una con el estado de la cola y la hora estimada de comienzo. Durante esta acción, será necesario compensar el valor del estado con los huecos libres existentes entre las citas, ya que éstos alivian la carga de trabajo del empleado reduciendo proporcionalmente el retraso. Para ello, se restará al valor del estado la diferencia en tiempo entre la hora programada de comienzo de la cita en la que se encuentre el cursor de la iteración, y la hora programada para la finalización de la cita anterior.

Hora de lanzamiento de la operación: 8:40				
Horario programado de cita en ejecución	**Hora de comienzo de cita en ejecución**	**Hora estimada de finalización de cita en ejecución**	**Estado de cola (minutos)**	
8:10 - 8:15	8:37	8:42	27	
Horario programado de la siguientes citas en la cola y dentro de los límites de la ventana de tiempo			**Estado de cola individual (minutos)**	**Hora estimada de comienzo de cita**
8:15 - 8:20			27	8:42
8:30 - 8:35			17	8:47
8:55 - 9:00			0	8:55

Tabla 50. Ejemplo de una operación de notificación de estado para el caso en que la cola de citas presente retraso respecto a lo programado.

Nótese que el estado de cola individual para la segunda cita sale de restar a los 27 minutos de retraso la diferencia entre las 8:30 y 8:20, dando como resultado 17 minutos. Igualmente ocurre en la tercera cita cuyo estado individual es de -3 minutos,

pero éste automáticamente pasa a valer 0, pues con el hueco libre entre ella y la anterior se compensa el retraso acumulado en la cola de citas.

Los mensajes enviados a cada cita mediante notificaciones, mostrarán textualmente los valores correspondientes con las dos últimas columnas de la Tabla 50.

CAPÍTULO 4: IMPLEMENTACIÓN

Este capítulo se centrará en detallar los aspectos más importantes y relevantes de la parte del código destinado a la implementación del módulo de control automático para el estado de las colas de citas.

4.1. Aspectos de la implementación del módulo de control

Como ya se comentó en anteriores capítulos, el módulo de control automático para el estado de las colas de citas comprende el cálculo dinámico de la cantidad en tiempo, de retraso o adelanto que pueda presentar cada una de las colas de citas, esto es, determinar el estado de las colas, y la notificación de dicho resultado a los usuarios correspondientes, mediante notificaciones directas a sus dispositivos móviles. Sin embargo, antes de que el módulo de control comience su ejecución es necesario primero generar una cola de citas.

Una cola de citas es generada cuando un administrador asigna a un empleado un turno de trabajo, lo que a nivel de código supone la creación y posterior guardado de un objeto de la clase *Shift.* Justo en el momento previo al guardado de dicho objeto en la base de datos se lanza un método denominado "*build_appointments",* cuya misión es generar automáticamente un grupo de citas preparadas para su reserva, y con un horario programado de comienzo y finalizado comprendido entre los límites de tiempo del nuevo turno de trabajo. En la siguiente pieza de código, se muestra un detalla de la correspondiente implementación.

```
1. has_many :appointments
2. belongs_to :workday

3. before_create :build_appointments

4. def build_appointments
5.     gap = self.workday.activity.service
6.     aux = self.starts_at
7.     begin
8.       self.appointments.build(:starts_at => aux,
                                  :ends_at => aux+gap.minutes,
                                  :state => 1, :alert => 0)
9.       aux += gap.minutes
10.   end while aux < self.ends_at
11. end
```

Código 1. Generación de una cola de citas en el fichero de modelo "Shift.rb".

La lista con la descripción de las principales acciones descritas en el Código 1 es:

1. Se establece una relación entre las clases *Shift* y *Appointments,* tal que una instancia perteneciente a la primera de ellas, tiene asociadas cero, una o varias instancias de la segunda.
2. Se establece una relación entre las clases *Shift* y *Workday*, tal que una instancia perteneciente a la primera de ellas, está asociada a otra instancia de la segunda.
3. Justo en el momento previo al guardado de un turno de trabajo, se lanza el método *build_appointments.*
4. Comienza el método *build_appointments.*
5. Se asigna a una variable *gap*, el valor del tiempo medio de cita de la actividad a la que pertenece el nuevo turno de trabajo.
6. Se asigna a una variable *aux,* la hora de comienzo del nuevo turno de trabajo.
7. Comienza un bucle.
8. Se crea una cita asociada al nuevo turno de trabajo con una hora programada de comienzo igual al valor de la variable *aux,* y de finalización con este último valor más el de la variable *gap*. Así se consigue generar citas con una duración exacta al valor medio de la actividad impuesto por el administrador. El estado de la cita se define como disponible.
9. Se incrementa la variable *aux* para que la siguiente cita a generar comienzo justo a la finalización de la última.
10. El bucle finaliza cuando el valor de la variable *aux* excede a la hora que termina el nuevo turno de trabajo.
11. Finaliza el método *build_appointments.*

Una vez la cola de citas ha sido creada, el siguiente paso es establecer una tarea programada que se encargue de realizar las dos operaciones básicas del modulo de control de forma periódica con la frecuencia suficiente para que los usuarios finales reciban el estado adecuadamente pero sin causar molestias (ver Apartado 4.1.1). La creación de una tarea supone generar un fichero de código *Ruby* con extensión "*rake*" en la carpeta del servidor con destino "*../lib/tasks".* En el siguiente código, se demuestra como se implementa esto.

```
File.open("#{Rails.root}/lib/tasks/queue#{@shift.id}.rake", "w") do |f|
  f.write("task :queue => :environment do")
  f.write("shift = Shift.find(@shift.id)")
  f.write("shift.launch_control!")
  f.write("end")
end
```

Código 2. Creación de una tarea para el módulo de control.

Notar en el código que el nombre del fichero con la tarea a ejecutar está compuesto por la palabra "*queue"* más el identificador único del nuevo turno de trabajo guardado con anterioridad. La tarea básicamente consiste en leer de la base de datos la correspondiente cola de citas, representada en el turno de trabajo, y lanzar un método de instancia de la clase *Shift*, denominado "*launch_control"*, el cual, ejecuta la operaciones básicas de cálculo y notificación de su estado. En la siguiente pieza de código se muestra la primera parte de dicho método, correspondiente a la primera de las operaciones básicas.

```
1.  def launch_control

2.    current_time = DateTime.now
3.    queue = self.appointments

4.    active_appointment = queue.select{|x| x.state == 4}.first
5.    next_appointment = queue.select{|x| x.state == 3}.select{|y| y.starts_at >= current_time}.first
6.    prev_appointment = queue.select{|x| x.state == 5}.first

7.    return if next_appointment.nil?

8.    if active_appointment.nil?
9.       state = (current_time.to_i - next_appointment.starts_at.to_i)/60
10.      state = 0 if (state < 0 && prev_appointment.nil?)
11.   else
12.      estimated_time = active_appointment.started_at + self.workday.activity.service_time.minutes
13.      if estimated_time > current_time
14.         state = (estimated_time.to_i - active_appointment.ends_at.to_i)/60
15.      else
16.         state = (current_time.to_i - active_appointment.ends_at.to_i)/60
17.      end
18.   end
19.   …
20. end
```

Código 3. Cálculo del estado de una cola de citas.

La descripción de cada línea es la siguiente:

1. Comienzo del método de instancia.
2. Se obtiene la fecha y hora exacta del momento en que se ejecuta el método, asignándolo a la variable *current_time*.
3. Se asigna a la variable *queue,* todas las citas asociadas a la instancia referente a un turno de trabajo, desde la que se ha lanzado el método. En resumen, se obtiene la cola de citas.
4. Se busca de entre la cola de citas, la primera cuyo estado sea de ejecución. Se almacena en la variable *active_appointment.*

5. Se busca de entre la cola de citas, la primera cuyo estado sea de reserva y confirmación en espera de ser ejecutada. Se almacena en la variable *next_appointment.*
6. Se busca de entre la cola de citas, la última que haya terminado su ejecución. Se almacena en la variable *prev_appointment.*
7. En caso de que la variable next_appointment esté vacía se cancelará el método. Esto ocurrirá cuando la cola de citas no tenga más citas en espera.
8. En caso de no existir ninguna cita en ejecución por parte de un empleado se pasará a las línea 9.
9. El valor del estado de la cola será la diferencia en minutos entre el tiempo de ejecución del método (*current_time)* y la hora programada de comienzo de la cita almacenada en la variable *next_appointment.* Notar que para obtener la diferencia en minutos es necesario convertir primeramente las fechas y horas a segundos mediante la instrucción *to_i,* y posteriormente dividir entre 60.
10. Cuando el valor del estado de cola obtenido durante la línea 9 sea menor que cero, es decir, presente adelanto, y no exista una última anteriormente ejecutada, esto es, la cola aún no ha comenzado, el valor de estado pasará a valer cero.
11. En caso de existir una cita en ejecución por parte de un empleado se pasará a la línea 12. Si no es así, se saltará a la 19.
12. Se determina el tiempo estimado de finalización de la cita en ejecución, mediante la suma de la hora en la que comenzó dicha cita y el tiempo medio asignado a la actividad que pertenece la instancia desde donde se invocó este método.
13. Si está previsto que la cita en ejecución cambie al siguiente estado de finalizado en un tiempo menor al asignado para la actividad se pasa a la línea de 14.
14. El valor del estado de la cola será la diferencia en minutos entre el tiempo estimado de finalización de la cita en ejecución y el programado.
15. De no cumplirse la condición de la línea 13, se pasa a la línea 16.
16. El valor del estado de la cola será la diferencia en minutos entre el tiempo de ejecución del método (*current_time)* y el programado para la finalización de la cita en curso.
17. Fin de la condición impuesta en la línea 13.
18. Fin de la condición impuesta en la línea 8.
19. Continuación del método con la notificación del valor del estado de cola, calculado en las líneas de código anteriores.
20. Fin del método de instancia.

Se puede observar como la estructura del código sigue las directrices marcadas por el diagrama de flujo de la Figura 45.

En la siguiente pieza de código, se muestra la continuación mencionada en la línea 19 del Código 3, esto es, la correspondiente notificación del valor del estado de cola a los usuarios finales con cita reservada en ella.

```
1.  if state < 0
2.    if active_appointment.nil?
3.       appointments_to_be_notified = queue.select{|x|
                    x.starts_at >= current_time}.select{|y|
                    y.starts_at < (current_time + 30.minutes)}.select{|z| z.state == 3}
4.    else
5.       appointments_to_be_notified = queue.select{|x|
                     x.starts_at >=  active_appointment.ends_at}.select{|y|
                    y.starts_at < (active_appointment.ends_at + 30.minutes)}.select{|z|
                    z.state == 3}
6.    end
7.    appointments_to_be_notified.each do |appointment|
8.       body_message = get_message(state,appointment)
9.       appointment.user.devices.each do |device|
10.         Notification.create!(:alert => body_message, :sound => true, :badge => 1, :device => device)
11.      end
12.   end
13. else
14.    if active_appointment.nil?
15.       appointments_to_be_notified = queue.select{|x|
                    x.starts_at >= current_time}.select{|y|
                    y.starts_at < (current_time + state + 30.minutes)}.select{|z| z.state == 3}
16.    else
17.       appointments_to_be_notified = queue.select{|x|
                    x.starts_at >= active_appointment.ends_at}.select{|y|
                    y.starts_at < (active_appointment.ends_at + state + 30.minutes)}.select{|z|
                    z.state == 3}
18.    end
19.    aux = prev_appointment.ends_at
20.    appointments_to_be_notified.each do |appointment|
21.       state += (appointment.starts_at.to_i - aux.to_i)/60
22.       state = 0 unless state > 0
23.       body_message = get_message(state,appointment)
24.       appointment.user.devices.each do |device|
25.          Notification.create!(:alert => body_message, :sound => true, :badge => 1, :device => device)
26.       end
27.    end
28. end
29. APN::Notification.send_notifications
```

Código 4. Notificación del estado.

La descripción de cada línea de código es la siguiente:

1. Si el signo del estado calculado con anterioridad es negativo (existe adelanto), se pasará al bloque de código comprendido entre las líneas 2 y 12.
2. Si no existe una cita en curso durante la ejecución del método de instancia *launch_control* se pasa a la línea 3.
3. Se almacena en un array de nombre *appointments_to_be_notified*, las citas reservadas en la cola cuya hora de comienzo programada se encuentre dentro

de los límites de una ventana de tiempo con tamaño igual a 30 minutos y origen en el momento de ejecución del método de instancia.
4. Si la condición de la línea 2 no se cumple, se pasa a la línea 5.
5. Se almacena en un array de nombre *appointments_to_be_notified*, las citas reservadas en la cola cuya hora de comienzo programada se encuentre dentro de los límites de una ventana de tiempo con tamaño igual a 30 minutos y origen en la hora programada para la finalización de la cita en curso.
6. Fin de la condición iniciada en la línea de código 2.
7. Se itera cita a cita sobre *appointments_to_be_notified*.
8. Se lanza un método auxiliar, denominado *get_message*, que construye la cadena de texto personalizada (ver Apartado 4.1.2), enviada en el cuerpo de la notificación al usuario. Se almacena en la variable *body_message.*
9. Se itera sobre cada dispositivo que posee el usuario asociado a la cita que se pretende avisar del estado.
10. Se crea una notificación en la base de datos a la espera de ser enviada.
11. Fin de la iteración sobre los dispositivos.
12. Fin de la iteración sobre las citas.
13. Si la condición de la línea 1 no se cumple, se pasa a ejecutar el bloque de líneas comprendido entre la 14 y 27.
14. Si no existe una cita en curso durante la ejecución del método de instancia *launch_control* se pasa a la línea 15.
15. Se almacena en un array de nombre *appointments_to_be_notified*, las citas reservadas en la cola cuya hora de comienzo programada se encuentre dentro de los límites de una ventana de tiempo con tamaño igual a la suma de una cantidad fija de 30 minutos y el valor del estado calculado anteriormente. El origen de dicha ventana se encuentra en el momento de ejecución del método de instancia.
16. Si la condición de la línea 14 no se cumple, se pasa a la línea 17.
17. Se almacena en un array de nombre *appointments_to_be_notified*, las citas reservadas en la cola cuya hora de comienzo programada se encuentre dentro de los límites de una ventana de tiempo con tamaño igual a la suma de una cantidad fija de 30 minutos y el valor del estado calculado anteriormente. El origen de dicha ventana se encuentra en la hora programada para la finalización de la cita en curso.
18. Fin de la condición iniciada en la línea 14.
19. Se almacena en la variable *aux,* la hora de finalización programada para la última cita ejecutada y terminada por un empleado.
20. Se itera cita a cita sobre *appointments_to_be_notified*.
21. Se compensa el valor del estado en caso de que exista huecos libres en la cola.
22. Si el valor del estado no presenta signo positivo, se igual a cero.
23. Se lanza un método auxiliar, denominado *get_message (*ver Apartado 4.1.2*)*, que construye la cadena de texto personalizada, enviada en el cuerpo de la notificación al usuario. Se almacena en la variable *body_message.*
24. Se itera sobre cada dispositivo que posee el usuario asociado a la cita que se pretende avisar del estado.
25. Se crea una notificación en la base de datos a la espera de ser enviada.
26. Fin de la iteración sobre los dispositivos.
27. Fin de la iteración sobre las citas.
28. Fin de la condición iniciada en la línea de código 1.
29. Se envían todas las notificaciones creadas en las líneas anteriores.

Cabe destacar, que al igual que en la pieza de Código 3, todas estas líneas siguen las pautas marcadas en la Sección 3.5, y en concreto para el caso de la notificación del estado, la referencia es el diagrama de flujo de la Figura 46.

4.1.1. Programador de tareas

Una aspecto muy importante del módulo de control automático de colas de citas será la utilización de un programador que sea capaz de ejecutar la tarea de cálculo y notificación, definida para cada cola de una manera ordenada y periódica. Dicho programador, se implementará con la ayuda de una librería llamada "*whenever"*, compatible con la tecnología Ruby on Rails, cuya misión es configurar la ejecución periódica de tareas en el servidor. Ver más detalles en [22].

En la pieza de Código 5, se detalla como hacer que una tarea sea lanzada en intervalos fijos de tiempo.

```
File.open("#{Rails.root}/config/schedule.rb", "a") do |f|
  f.write("every @activity.service_time.minutes do")
  f.write("rake 'queue#{@shift.id}'.to_sym:queue")
  f.write("end")
end
```

Código 5. Activación de una tarea en el programador de tareas.

El código simplemente abre el fichero de configuración "*schedule.rb",* que es creado durante la instalación del programador de tareas, y define la periodicidad de un proceso encargado de lanzar la tarea definida, en este caso por el Código 2, en intervalos iguales a la duración media de cada cita establecida por un administrador para la actividad en la que se encuentre la cola o turno de trabajo sobre la que actuará la tarea.

A continuación, es necesario reflejar la escritura realizada en el fichero sobre el programador de tareas para activar el nuevo proceso.

```
whenever --update-crontab
```

Código 6. Actualización del programador de tareas.

4.1.2. Constructor de mensajes

El cuerpo del mensaje incluido en cada notificación enviada al usuario, contendrá en los casos de adelanto o retraso: el valor del estado de la cola, la hora de comienzo programada de la cita, y su hora estimada de comienzo. Por ejemplo, si el estado de una cola de citas presenta un retraso de 27 minutos a las 8:40 y se desea avisar a la cita programada para su comienzo a las 8:15, el mensaje enviado sería:

"Su cita programada para las 8:15 presenta un retraso de 27 minutos. Se estima que comience a las 8:42"

La parte del código encargada de construir el texto de un mensaje se ha desviado a un método auxiliar, denominado "*get_message",* para hacerlo más legible (ver Códigos 3 y 4). Este método recibe dos parámetros fundamentales para la personalización del mensaje: el valor del estado y la cita a notificar. En la pieza de Código 7, se detalla su implementación.

```
def get_message(q,app)
 if q > 0
   If q >= 60
     q = (q/60).to_i
     return "Su cita programada para las #{app.starts_at} presenta un retraso de #{q} horas.
                Se estima que comience a las
                #{(app.starts_at + q.minutes).strftime('%H:%M')}".force_encoding("UTF-8")
   else
     return "Su cita programada para las #{app.starts_at} presenta un retraso de #{q} minutos.
                Se estima que comience a las
                #{(app.starts_at + q.minutes).strftime('%H:%M')}".force_encoding("UTF-8")
 elsif q < 0
   if q <= -60
     q = (q/60).to_i
     return "Su cita programada para las #{app.starts_at} presenta un adelanto de #{-q} horas.
                Se estima que comience a las
                #{(app.starts_at + q.minutes).strftime('%H:%M')}".force_encoding("UTF-8")
   else
     return "Su cita programada para las #{app.starts_at} presenta un adelanto
                de #{-q} minutos. Se estima que comience a las
                #{(app.starts_at + q.minutes).strftime('%H:%M')}".force_encoding("UTF-8")
 else
     return "Su cita programada para las #{app.starts_at} está prevista que comience a su
                hora".force_encoding("UTF-8")
 end
end
```

Código 7. Construcción del mensaje incluido en una notificación.

Como se puede observar, el método *get_message* devuelve una cadena de texto personalizada según el valor y signo del estado. Además, en los casos de adelanto y retraso, cuando el valor del estado es igual o mayor a 60 minutos, éste se aproxima a unidades de hora para hacer más claro el mensaje a los usuarios finales. Notar también que todas las cadenas de texto están codificadas a Unicode Transformation Format-8 (UTF-8), con lo que se evita problemas en su visualización cuando presenten acentos o caracteres especiales.

4.2. Resultado de la implementación del módulo de control

El producto de salida del módulo de control automático para colas de citas, es la notificación que llega al usuario informando del retraso o adelanto en la cola donde ha reservado con anterioridad una cita. En la Figura 47, se puede observar el resultado real de dicho producto.

Figura 47. Visualización en pantalla de una notificación informando del estado de una cola de citas y la correspondiente estimación del comienzo de la cita reservada por el usuario final.

Cuando el dispositivo móvil, en este caso un iPhone, recibe la notificación enviada por el módulo de control del servidor, despierta de su estado de reposo alertando al usuario final de dicho evento. En todo momento, éste último estará informado del progreso de la cola de manera que no tenga que esperar en el lugar de la cita cuando exista un retraso, o por el contrario, llegue tarde porque la cola va con adelanto.

Notar la inclusión en la pantalla de la Figura 47, de un deslizador en la parte inferior con el texto "*ver*". Si el usuario final lo desliza con el dedo hacia la derecha, la aplicación móvil de servicio se lanza automáticamente permitiendo a éste, realizar diversas acciones como: comprobación, reserva o cancelación de citas, modificación de

sus datos personales en el registro del sistema, visualización de la ruta en coche hasta el lugar de una actividad, etc… En las siguientes figuras, se mostrará como se visualizan varias de estas acciones en la aplicación móvil.

Figura 48. Visualización en pantalla de la lista de citas reservas por el usuario final.

Figura 49. Visualización en pantalla de una reserva y confirmación de cita para una determinada actividad.

CAPÍTULO 5: GESTIÓN DEL PROYECTO

5.1. Planificación

En esta sección se detallará en profundidad la metodología de trabajo llevada a cabo durante todo el proyecto. Se comenzará explicando una planificación inicial realizada en los primeros momentos del proyecto para finalmente comparar con la planificación real que tuvo lugar durante la ejecución del mismo. Ambas planificaciones estarán apoyadas en diagramas de Gantt como resumen gráfico y visual de su detalle.

5.1.1. Planificación inicial

Recordando lo que sé comentó en la Sección 1.1, este proyecto propone un sistema automatizado de notificaciones a usuarios, con información detallada de cuándo comenzarán sus citas previamente reservadas en cualquiera de los servicios ofertados por un cliente. Ante esta proposición, se decidió por una planificación inicial que se ajustase a las fases comúnmente utilizadas para la puesta en marcha y desarrollo de un proyecto de automatización:

Figura 50. Fases en un proyecto de automatización.

Tal decisión estuvo motivada en gran medida por la definición que aplica la Real Academia de las Ciencias Físicas y Exactas al termino *automatización*[32]*:*

> *"La automatización es la aplicación del conjunto de métodos y procedimientos para la substitución del operario en tareas físicas y mentales previamente programadas, al control de procesos industriales".*

Aplicando cierta analogía con el sistema proyectado aquí, se puede relacionar a los métodos y procedimientos para la substitución del operario con las herramientas de software utilizadas durante la implementación, y a las tareas físicas y mentales previamente programadas con la operación de aviso al usuario. Además, el proceso industrial[33] en este caso, sería la transformación del estado de una cola de citas (ya sea un adelanto, retraso o a tiempo) en un grupo de notificaciones para los usuarios.

---

[32] Ver [21].

[33] Proceso industrial: "*parte del sistema en que, a partir de la entrada de material, energía e información, se genera una transformación sujeta a perturbaciones del entorno, que da lugar a la salida de material en forma de producto*". Ver [21].

En los siguientes apartados se dará explicación a las diferentes tareas contempladas dentro de la planificación ideal y agrupadas según las fases mostradas en la Figura 50.

5.1.1.1. Automatización

Esta primera fase de automatización comprende una observación del proceso (control del estado de cada cola de citas y posterior notificado) que se pretende automatizar, valga la redundancia, y de las tecnologías necesarias para poder llevarlo a cabo. Las actuaciones previstas durante esta fase son:

- Observación teórica del proceso a controlar.
- Diagrama de estados general del proceso.
- Análisis de la tecnología a utilizar.
- Estudio teórico de procesos similares ya existentes en el mercado.

5.1.1.2. Supervisión

En la segunda fase, denominada de supervisión, es necesario profundizar a un nivel más bajo sobre el diagrama de estados general del proceso. Para ello, se previó las siguientes actuaciones:

- Modulación del diagrama de estados general.
- Validación del diagrama de estados general y de sus módulos.

La primera supone dividir, en la medida de lo posible, el diagrama de estados general en módulos más pequeños, que ayuden a su implementación y prueba. La segunda actuación supone "*supervisar*" la evolución del proceso dentro del diagrama de estados.

5.1.1.3. Interacción

Por fase de interacción en un proceso de automatización industrial se entiende básicamente por el diseño en papel de mando del operario, esto es, cómo va a interactuar este último con el proceso a controlar. En lo que atañe a este proyecto, se puede decir que para esta fase son necesarias las siguientes actuaciones:

- Diseño de la aplicación web de gestión para los perfiles de administrador y empleado.
- Diseño de la aplicación móvil de servicio para los perfiles de usuario final.

5.1.1.4. Implementación

Como indica su nombre, en la cuarta etapa se lleva a cabo la implementación de todo el sistema, esto es, la traducción del diagrama de estados y del diseño de las diferentes aplicaciones a un lenguaje de programación.

5.1.1.5. Pruebas

Una vez terminada la fase de implementación solamente quedaría hacer las correspondientes pruebas. La metodología a seguir en este caso es, intentar depurar el código en el módulo donde se esté ocasionando un error, y si éste persigue, recorrer nuevamente las fases de arriba hacia abajo.

5.1.1.6. Diagrama de Gantt

En la Figura 51, se muestra la previsión en el tiempo de cada una de las fases comentadas en los apartados anteriores. Notar que el proyecto estaba previsto que comenzase el 7 de Julio de 2011 para finalizar el 2 de Mayo de 2012.

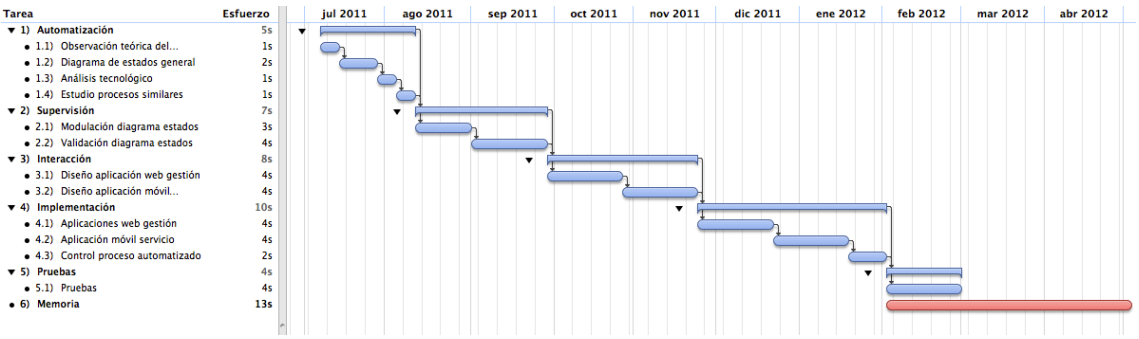

Figura 51. Diagrama de Gantt: planificación inicial.

5.1.2. Planificación real

La planificación inicial sufrió un ligero adelanto durante las tres primeras fases del proyecto para después retrasarse durante la etapa de implementación. El desarrollo de varias aplicaciones de gestión y de servicio en dos plataformas diferentes, provocó la necesidad de leer exhaustivamente la extensa documentación de estas últimas. No ayudaron tampoco ciertas mejoras en la usabilidad e interfaz de las aplicaciones, que no se han creído necesarias de detallar en el siguiente diagrama de Gantt.

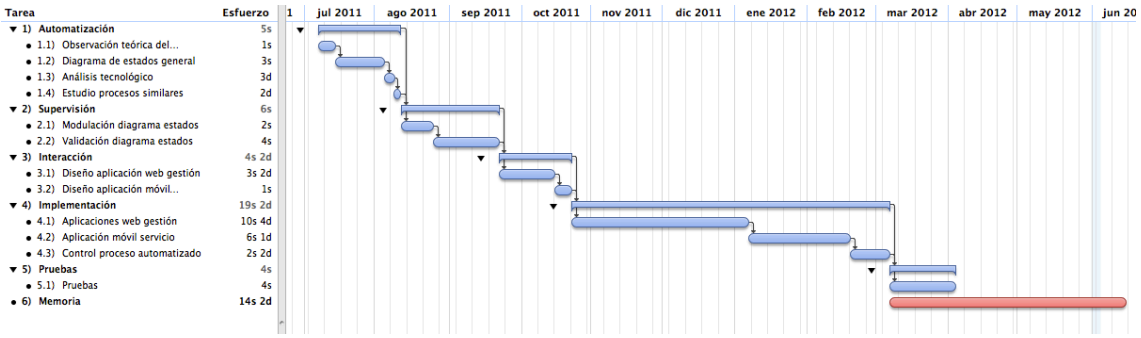

Figura 52. Diagrama de Gantt: planificación real.

El proyecto comenzó puntual con la planificación inicial, es decir, el 7 de Julio de 2011. Por el contrario, éste acabó el 14 de Junio de 2012 con aproximadamente un mes de retraso total sobre la fecha final estimada.

5.2. Presupuesto

En esta sección se procederá a detallar los costes que suponen cada una de las partes del proyecto, esto es, personal, hardware y software. Para ello, se comparará un presupuesto inicial estimado en los primeros momentos del proyecto, con un presupuesto final que contempla el coste real de todas las actuaciones una vez finalizado.

5.2.1. Presupuesto inicial estimado

Para conseguir una estimación lo más precisa posible se ha intentado dividir los posibles costes en cuatro grandes grupos: personal, hardware, software e indirectos. Durante los siguientes apartados se pasará a detallar cada uno de ellos.

5.2.1.1. Costes estimados de personal

Los costes estimados de personal, en este caso, se componen únicamente de los honorarios del Ingeniero Industrial que desarrolla el proyecto. Dicha cuantía es fácilmente calculable con la cantidad que estipula el Colegio Oficial de Ingenieros Industriales de Madrid[34]. Para el computo de horas totales trabajadas por el ingeniero, se considera una jornada laboral media de 7 horas diarias y 35 semanales durante el total de meses contemplados en la planificación inicial del Apartado 5.1.1.6, esto es aproximadamente unos 10 meses o 42 semanas.

[34] Ver en [34].

ud.	Concepto	Cantidad	Precio Unitario	Precio Total
horas	Ingeniero Industrial	1470	€ 40,00	€ 58.800,00
				€ 58.800,00

Tabla 51. Costes estimados de personal.

5.2.1.2. Costes estimados de hardware

El coste estimado de hardware hace referencia a aquél causado por la adquisición de todos los equipos informáticos necesarios para la realización del proyecto. Se tendrá en cuenta su amortización durante toda la duración de este último.

ud.	Concepto	Cantidad	Precio Unitario	Vida útil	Tiempo de uso	Precio Total
-	Portátil Apple Macbook Pro 13”	1	€ 1.145,00	60 meses	10 meses	€ 190,83
-	SmartPhone Apple iPhone 4S 16Gb	1	€ 599,00	24 meses	10 meses	€ 249,58
						€ 440,41

Tabla 52. Costes estimados de hardware.

Nótese que no ha sido incluido como pieza de hardware un servidor compatible con la tecnología Ruby on Rails. Esto se debe a que durante la etapa de implementación el portátil que aparece en la primera fila ha funcionado como servidor local de pruebas.

5.2.1.3. Costes estimados de software

El coste estimado de software supone la cantidad necesaria a pagar por las licencias de cada uno de los programas informáticos utilizados en este proyecto. Comprende software de muy diferentes ámbitos: programación web y móvil, edición fotográfica, planificación, etc.. Al igual que en el caso del hardware, es necesario contemplar su amortización.

ud.	Concepto	Cantidad	Precio Unitario	Vida útil	Tiempo de uso	Precio Total
-	Apple Mac OS X Lion	1	€ 23,99	24 meses	10 meses	€ 9,99
-	OmniPlan 1.7	1	€ 149,99	24 meses	10 meses	€ 62,99
-	OmniGraffle Pro 5.4	1	€ 149,99	24 meses	10 meses	€ 62,99
-	Pixelmator 2	1	€ 23,99	24 meses	10 meses	€ 9,99
-	Sublime Text Editor 2	1	€ 43,70	24 meses	10 meses	€ 18,21
-	Espresso 2.2	1	€ 55,00	24 meses	10 meses	€ 22,92
-	SDK iOS (Xcode, Instruments,..)	1	€ 0,00	12 meses	10 meses	€ 0,00
-	Apple iPhone Developer Program	1	€ 79,00	12 meses	10 meses	€ 65,83
						€ 252,92

Tabla 53. Costes estimados de software.

Nótese que el concepto Apple iPhone Developer Program no representa en sí una pieza de software. Se trata de una cuota anual que hay que pagar a Apple para poder testear aplicaciones en dispositivos reales y publicarlas en el App Store.

5.2.1.4. Costes estimados indirectos

Para estimar el total del coste indirecto del proyecto se ha considerado que este último ha sido realizado íntegramente desde el domicilio del ingeniero, por lo que sólo habrá que tener en cuenta el consumo de luz de los equipos informáticos y la factura telefónica del proveedor de Internet.

ud.	Concepto	Cantidad	Precio Mensual	Tiempo de uso	Precio Total
-	Luz	-	€ 50,00	10 meses	€ 500,00
-	Internet	-	€ 42,00	10 meses	€ 420,00
					€ 920,00

Tabla 54. Costes estimados indirectos.

Nótese que ningún otro proyecto ha sido desarrollado durante el período de amortización.

5.2.1.5. Total coste estimado

En la siguiente tabla se muestra el resumen del presupuesto inicial del proyecto con la cantidad total estimada, que no es más que la suma de lo calculado en apartados anteriores.

Grupo	Coste
Personal	€ 58.800,00
Hardware	€ 440,41
Software	€ 252,92
Indirectos	€ 920,00
Subtotal	€ 60.413,33
Margen de beneficio (20%)	€ 12.082,66
Margen de riesgo (10%)	€ 6.041,33
	€ 78.537,32

Tabla 55. Resumen presupuesto inicial estimado.

Según la Tabla 55, el proyecto tendría un coste inicial de **78.537,32 €**, incluidos los márgenes de beneficio y de riesgo.

5.2.2. Presupuesto final

Una vez finalizadas todas las actuaciones previstas es momento de calcular el coste real de todo el proyecto y comparar si se ajusta a lo estimado en el presupuesto inicial. Para que dicha comparativa sea coherente, los costes reales se separarán en los mismos cuatro grupos anteriores.

5.2.2.1. Costes reales de personal

A lo largo del desarrollo del proyecto, el ingeniero práctico una jornada media inferior a lo estimado en la planificación inicial, pero de igual cantidad por semana. En concreto, éste dedicó 5 horas diarias de trabajo por 35 semanales, compaginando el proyecto con prácticas curriculares en una empresa externa. La duración total fue aproximadamente de 11 meses o 49 semanas (ver Apartado 5.1.2).

ud.	Concepto	Cantidad	Precio Unitario	Precio Total
horas	Ingeniero Industrial	1715	€ 40,00	€ 68.600,00
				€ 68.600,00

Tabla 56. Costes reales de personal.

5.2.2.2. Costes reales de hardware

En el caso de los costes reales de hardware, software e indirectos ocurre de la misma manera que en el de personal, aumentan significativamente por el desvío en la duración estimada del proyecto.

ud.	Concepto	Cantidad	Precio Unitario	Vida útil	Tiempo de uso	Precio Total
-	Portátil Apple Macbook Pro 13”	1	€ 1.145,00	60 meses	11 meses	€ 209,92
-	SmartPhone Apple iPhone 4S 16Gb	1	€ 599,00	24 meses	11 meses	€ 274,54
						€ 484,46

Tabla 57. Costes reales de hardware.

5.2.2.3. Costes reales de software

ud.	Concepto	Cantidad	Precio Unitario	Vida útil	Tiempo de uso	Precio Total
-	Apple Mac OS X Lion	1	€ 23,99	24 meses	11 meses	€ 10,99
-	OmniPlan 1.7	1	€ 149,99	24 meses	11 meses	€ 68,74
-	OmniGraffle Pro 5.4	1	€ 149,99	24 meses	11 meses	€ 68,74
-	Pixelmator 2	1	€ 23,99	24 meses	11 meses	€ 10,99
-	Sublime Text Editor 2	1	€ 43,70	24 meses	11 meses	€ 20,03
-	Espresso 2.2	1	€ 55,00	24 meses	11 meses	€ 25,21
-	SDK iOS (Xcode, Instruments,..)	1	€ 0,00	12 meses	11 meses	€ 0,00
-	Apple iPhone Developer Program	1	€ 79,00	12 meses	11 meses	€ 72,42
						€ 277,12

Tabla 58. Costes reales de software.

5.2.2.4. Costes reales indirectos

ud.	Concepto	Cantidad	Precio Mensual	Tiempo de uso	Precio Total
-	Luz	-	€ 50,00	11 meses	€ 550,00
-	Internet	-	€ 42,00	11 meses	€ 462,00
					€ 1012,00

Tabla 59. Costes reales indirectos.

5.2.2.5. Total coste real

En la siguiente tabla se resume el presupuesto final con su correspondiente total, esto es, la suma de los totales obtenidos en cada grupo de costes reales.

Grupo	Coste final real	Coste inicial estimado
Personal	€ 68.600,00	€ 58.800,00
Hardware	€ 484,46	€ 440,41
Software	€ 277,12	€ 252,92
Indirectos	€ 1012,00	€ 920,00
Subtotal	€ 70.373,58	€ 60.413,33
Margen de beneficio (20%)	-	€ 12.082,66
Margen de riesgo (10%)	-	€ 6.041,33
	€ 70.373,58	€ 78.537,32

Tabla 60. Resumen presupuesto final real.

El coste final del proyecto asciende a **70.373,58 €**, lo que supone una disminución de **8.163,74 €** respecto al inicial. Sin embargo, esta diferencia se debe a la inclusión en el coste estimado de un margen de beneficio y otro de riego cuyas cantidades suponen **12.082,66 €** y **6.041,33 €**, respectivamente. Se puede decir entonces, que el margen de riesgo se cobraría íntegro, además de una pequeña fracción del beneficio que rondaría los **2.122,41 €.**

5.3. Estrategia comercial

Una vez conocido el coste real del proyecto, es el momento de determinar la posterior explotación de éste. Se ha decidido plantear varios modelos de negocio, que definan una futura estrategia comercial, como eje principal de un posible lanzamiento al mercado del sistema, mediante la integración de una suite de aplicaciones web para el cliente, una aplicación móvil para usuarios finales y un servicio de mantenimiento del servidor.

	Modelo A	Modelo B	Modelo C
Hosting de pago	SÍ	NO	NO
Aplicaciones web de pago	SÍ		
Aplicación móvil de pago	NO	SÍ	NO
Publicidad + Explotación datos de usuarios	NO	NO	Sí

Tabla 61. Modelos de negocio.

Con cada modelo de negocio se intenta cubrir diferentes requerimientos de los clientes prestando atención en parámetros como el volumen de usuarios finales, la necesidad de su servicio o la cantidad de empleados.

A continuación se detallarán de manera individual todos los modelos de negocio de la Tabla 61.

5.3.1. Modelo de negocio A

El primer modelo de negocio contemplaría cobrar un alquiler al cliente por el uso y mantenimiento de un servidor (*hosting*) propiedad de los autores de este proyecto, donde se establecería el sistema y almacenarían todos los datos. La cantidad a pagar por el cliente sería proporcional al volumen de datos y tráfico que soportase el servidor.

Las aplicaciones de gestión dirigidas al cliente nunca serían gratuitas. Su coste dependerá en gran medida de si éste requiere de más funcionalidades que no han sido contempladas en este proyecto y es necesario implementarlas. En caso de que no fuera así, se establecería un precio según los costes de desarrollo de cada aplicación añadiendo siempre un margen de beneficio.

Por otro lado, la aplicación móvil de servicio siempre se distribuiría a través del App Store[35] (igualmente para los demás modelos de negocio) y no supondría coste alguno para los usuarios finales que la adquiriesen en dicho medio. Tampoco existiría publicidad dentro de la aplicación ni se explotarían los datos de los usuarios finales recogidos en la base de datos.

[35] Ver Apartado 2.3.4

Este modelo de negocio está dirigido a clientes cuyo servicio no presenta un volumen elevado de clientes y desean una solución sencilla y rápida sin necesidad de buscar servidores propios.

5.3.2. Modelo de negocio B

En un segundo modelo, el cliente dispondría de uno o varios servidores gratuitos (mantenimiento incluido), corriendo con los costes los autores de este proyecto.

Al igual que en el modelo anterior, las aplicaciones de gestión tendrían un coste variable para el cliente en función de los requerimientos.

La mayor parte de los beneficios vendrían de la aplicación móvil dirigida a los usuarios finales, y cuyo coste no debería exceder de los 1,59 €[36]. Si así fuese, sus ventas no serían demasiado altas al tratarse de una aplicación con un uso regular de no más de varios minutos, por lo que es casi seguro que los usuarios finales no estarían dispuesto a pagar más de dicha cantidad por un producto de tales características. Es más razonable establecer un precio bajo y asequible para cualquiera que eleven la ventas.

Queda totalmente descartada la posibilidad de incluir publicidad y explotación de datos de usuarios pues se podría incurrir en quejas por parte de éstos al haber ya pagado un precio por la aplicación.

Es un modelo dirigido a un cliente con un volumen medio de usuarios finales, que oferte un servicio necesario para estos últimos (de ahí la obligación de pagar por adquirir la aplicación móvil).

5.3.3. Modelo de negocio C

Por último, se propone un modelo de negocio en el que al igual que el anterior, el alquiler y mantenimiento de los servidores no presentan coste alguno. De igual manera, esto es aplicable a la aplicación móvil de servicio para los usuarios finales.

Los beneficios vendrían del coste de las aplicaciones web de gestión para los empleados y administradores del cliente, y de la publicidad (ver Apartado 2.3.5) insertada en la aplicación móvil junto a la explotación[37] de los datos de usuarios finales alojados en la base de datos.

Este modelo va encaminado a clientes con un volumen de usuarios finales muy elevado, que permita obtener grandes beneficios en los casos explicados en el párrafo anterior.

---

[36] Tabla de precios en el App Store de Apple [10].

[37] Necesidad de autorización previa.

Los modelos de negocio aquí presentados no son en absoluto definitivos. Si las necesidades de los clientes no fuesen las aquí expuestas, sería necesario replantear nuevas posibilidades o incluso modelos adicionales, para flexibilizar en la medida de lo posible la estrategia comercial.

CAPÍTULO 6: PLAN DE PRUEBAS

En este capítulo se mostrarán los resultados obtenidos en varias pruebas ejecutadas sobre el módulo de control automático para colas de citas. Previo a esto, se detallará un simulador desarrollado para la realización de dichas pruebas.

6.1. Simulador de colas de citas

Durante el diseño e implementación del módulo de control automático para colas de citas, surgió el problema de no disponer de un medio físico en el que realizar las pruebas pertinentes. Ante esta situación, se decidió por crear un pequeño simulador de colas de citas que ayudase a comprobar si el diseño era el adecuado.

Básicamente, el simulador proporciona una interfaz de usuario con la que generar colas de citas aleatorias y visualizar el resultado de aplicar el algoritmo implementado para el módulo de control. Dicha interfaz ha sido creada en forma de aplicación móvil para un dispositivo iPad (ver Figura 53). Todo el código referente al módulo de control se encuentra en el servidor. Desde la aplicación únicamente se envía la orden de crear y simular una cola de citas para después recibir la información devuelta por el módulo de control y mostrarla en pantalla.

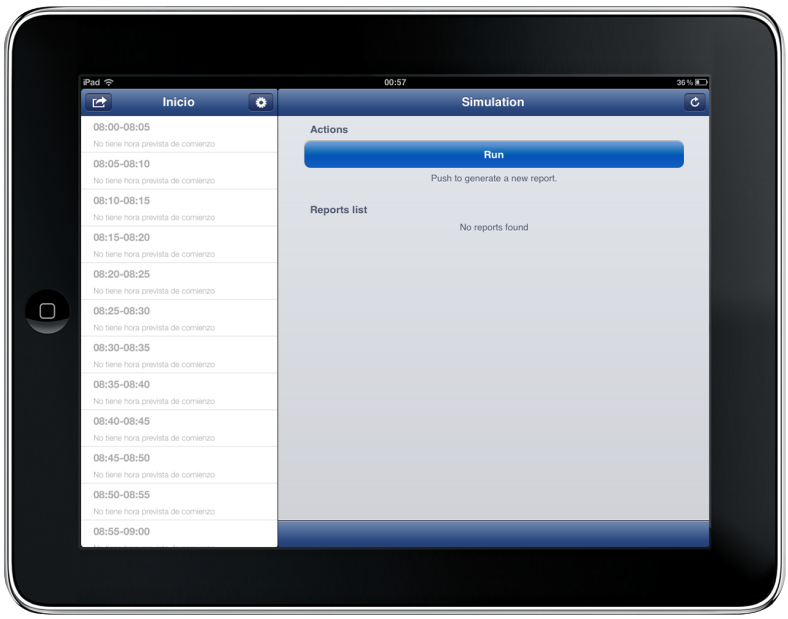

Figura 53. Vista general del simulador de colas de citas.

Como se puede observar en la Figura 53, la vista del simulador está dividida en dos secciones: maestra, y de detalle. Se han dado estos nombres a cada una de ellas, porque a nivel de código, la aplicación está gestionada por una instancia de la clase

UISplitViewController[38], propia del SDK en iOS, la cual presenta una interfaz "*master-detail*".

La sección maestra, situada en la parte izquierda de la pantalla, muestra una cola de citas vacía en forma de tabla. Para generar de manera automática y aleatoria esta última, sólo hay que seleccionar aquellos horarios en los que se desee que exista una cita reservada, y después pulsar el botón situado a la izquierda de la barra de navegación. Inmediatamente aparecerá en pantalla un menú contextual (ver Figura 54) con dos opciones: "*setup selection*" y "*clean*". La primera confirma la selección anterior y genera una nueva cola de citas, mientras que la segunda realiza la operación contraria, es decir, borra la cola.

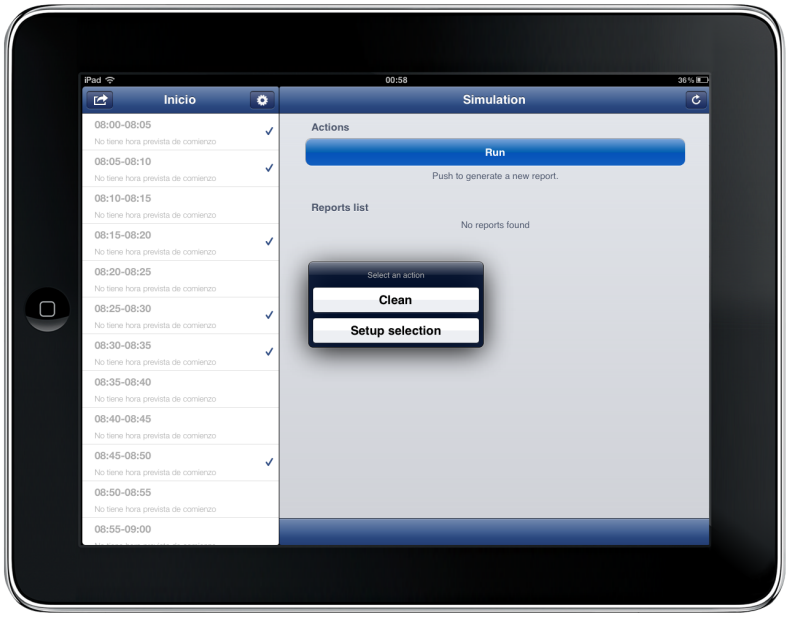

Figura 54. Menú contextual en la sección maestra.

Una vez es pulsada la opción "*setup selection*", el servidor recibe la orden y ejecuta las instrucciones necesarias para generar la nueva cola de citas. Cuando ha terminado, devuelve el resultado a la aplicación, y ésta refresca la tabla de la sección maestra (Ver Figura 55).

[38] Ver [23].

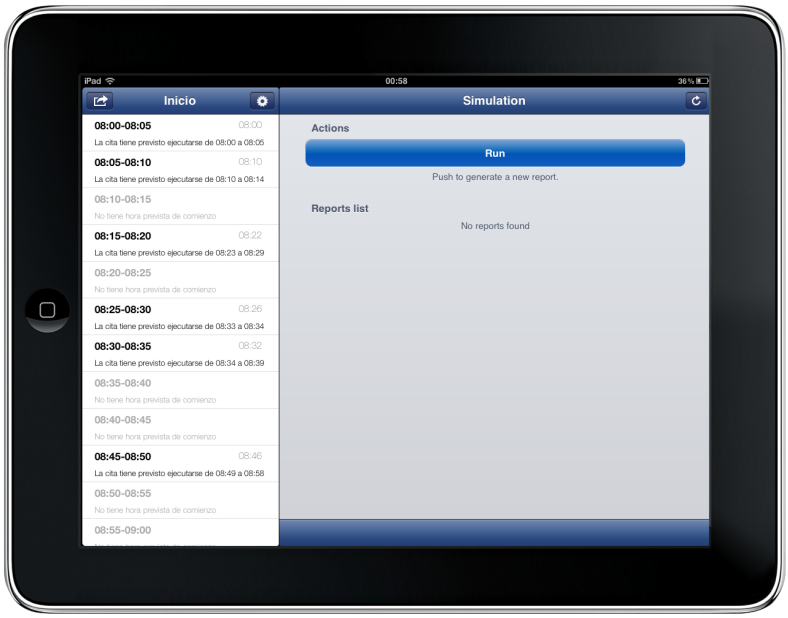

Figura 55. Vista general del simulador con una cola de citas aleatoria.

Nótese que cada cita representada en una celda de la tabla contenida en la sección maestra, contiene los siguientes parámetros:

- ➡ Hora programada de comienzo y finalización (texto de mayor tamaño y en negrita).
- ➡ Hora de llegada del usuario final (texto en color gris y más fino).
- ➡ Hora de comienzo y duración para la ejecución real de la cita (línea de texto en la parte inferior).

Los dos últimos parámetros son los que conforman la creación aleatoria de la cola. El algoritmo aplicado para ello, considera un proceso estocástico con comportamiento no-determinístico, es decir, el comportamiento del sistema se determina tanto por acciones predecibles del proceso, como por elementos aleatorios. Para este caso, estos últimos se corresponden con la llegada del usuario final al lugar de la cita y la duración de la misma cuando es ejecutada por el empleado a servicio del usuario. Su aleatoriedad se define mediante una distribución de probabilidad de Poisson, lo que permitirá obtener valores aleatorios cercanos a una frecuencia media de ocurrencia en dichos eventos. Ese valor de frecuencia podrá ser modificado desde un menú de ajustes, que aparecerá cuando el botón situado a la derecha de la barra de navegación de la sección maestra sea pulsado (ver Figura 56).

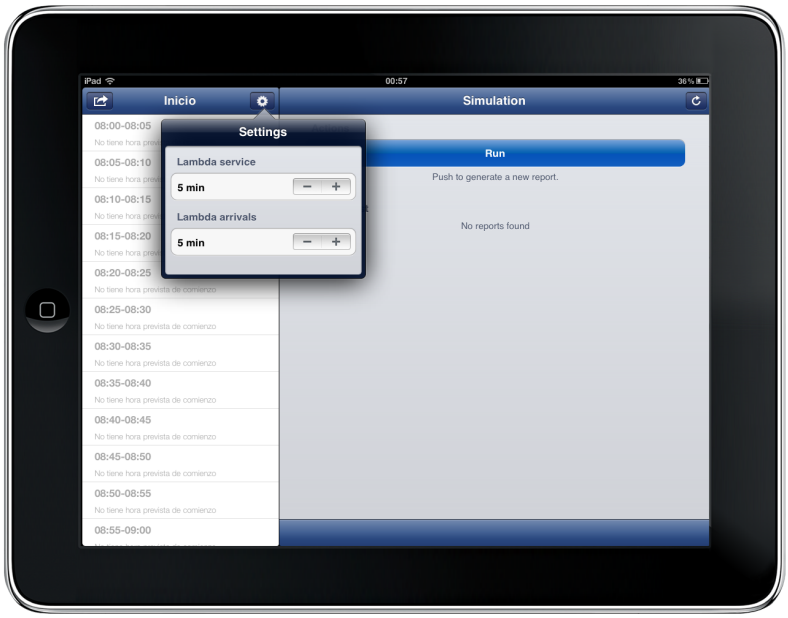

Figura 56. Menú de ajuste para la frecuencia de ocurrencia de eventos.

Como se puede observar, el menú de ajuste permite establecer la frecuencia media de ocurrencia en el caso de la duración de cada cita y en la llegada de los usuarios finales. En el ejemplo, ambos eventos presentan una media de 5 minutos. Si éste valor fuese mayor, se tendería a retrasar la cola de citas pues los usuarios finales tardarían más en llegar y los empleados necesitarían más tiempo de lo programado para ejecutar el servicio. De no ser así, es decir, el valor impuesto para cada evento fuese pequeño, se tendría a adelantar el flujo de la cola. Los usuarios finales llegarían con antelación y los empleados ejecutarían más rápidamente las citas. En resumen, este menú de ajuste ayuda a crear diferentes escenarios con los que corroborar que el módulo de control está correctamente implementado.

Cuando ya exista una cola de citas generada en la sección maestra, será el momento de proceder a su simulación. En la parte derecha de la pantalla (ver Figura 55), se encuentra la sección de detalle, que contiene el botón de lanzamiento de la simulación (*Run)*, una lista de informes con cada resultado y un botón de recarga para refrescar dicha lista. Una vez sea pulsado el botón de lanzamiento, el servidor realiza la correspondiente simulación sobre la cola establecida en la sección maestra, y devuelve a la aplicación el contenido de un informe con los resultados (ver Figura 57).

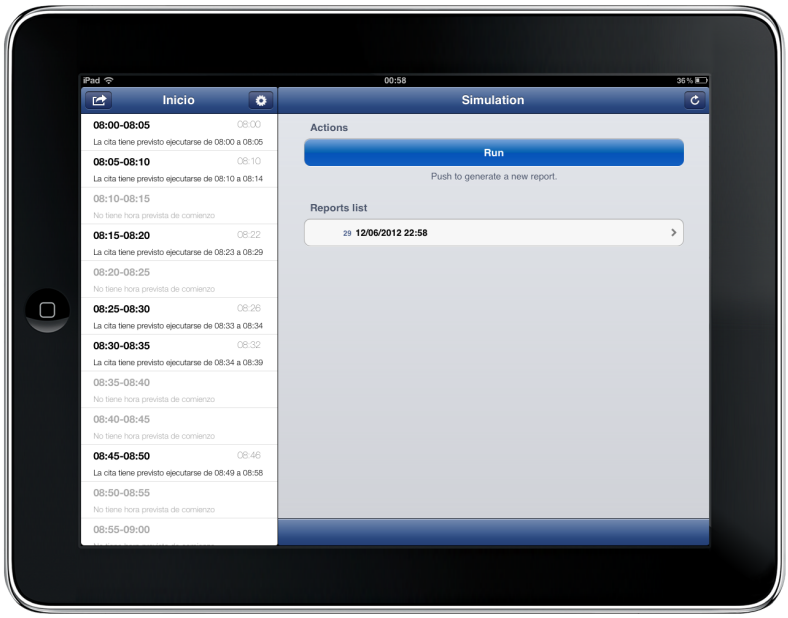

Figura 57. Vista general tras la simulación sobre una cola de citas.

Notar que el nuevo informe generado es mostrado en una lista con forma de tabla, debajo del botón de lanzamiento. Si se desea ver los resultados del mismo, sólo es necesario pulsar sobre la celda correspondiente, con lo que inmediatamente aparecerá una nueva vista sobre la sección de detalle con los resultados (ver Figura 58).

Figura 58. Vista general de los resultados de simulación sobre una cola de citas (ordenados por cita).

Los resultados mostrados en la Figura 58, detallan el cuerpo del mensaje (letra negrita) y la hora de llegada (letra azul claro), de todas las notificaciones que enviaría el módulo de control para cada cita reservada en la cola generada dentro de la sección maestra.

Si no se desease comprobar qué notificaciones llegarían a cada cita, sino cuáles de ellas serían enviadas por el módulo de control cada vez que éste es ejecutado en intervalos fijos de tiempo (ver Figura 59), sólo hay que pulsar el selector incluido en la barra de tareas de la parte inferior de la sección de detalle (ver Figura 58).

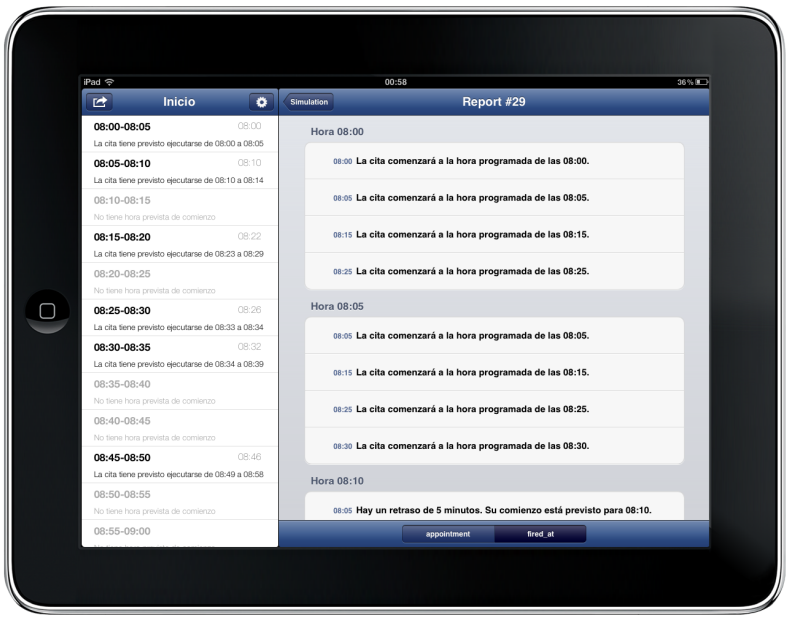

Figura 59. Vista general de los resultados de simulación sobre una cola de citas (ordenados por tiempo).

Como se puede observar en la Figura 59, al cambiar el selector, los resultados son ordenados tal que las notificaciones que enviaría el módulo de control en un caso real, son agrupadas por hora de envío. En cada una de ellas, se describe el cuerpo del mensaje (letra negrita) y la hora programada de comienzo de la cita a la que es enviada (letra azul claro).

A continuación, se demostrará el funcionamiento del simulador con varias pruebas realizadas a la finalización de la etapa de implementación.

6.2. Pruebas

La primera prueba realizada con el simulador ha sido una cola de citas reservadas con una duración programada de 5 minutos cada una. Para comprobar como se comporta el sistema ante un posible retraso en la cola se ha decidido asumir un valor medio de

probabilidad de 10 minutos para las llegadas de los usuarios finales y la duración real de cada cita. Los resultados se pueden visualizar en la siguiente tabla.

	Llegada del usuario	Comienzo de la cita	Finalizado de la cita	Hora estimada de la última notificación
08:00 - 08:05	08:00	08:00	08:15	08:00
08:05 - 08:10	08:08	08:15	08:28	08:15
08:10 - 08:15	08:16	08:28	08:35	08:25
08:20 - 08:25	08:35	08:35	08:44	08:35
08:25 - 08:30	08:41	08:44	08:51	08:40
08:35 - 08:40	09:02	09:02	09:11	09:00

Tabla 62. Cola de citas generada aleatoriamente con retraso.

Como se puede observar, los usuarios finales llegan con cierto retraso y las citas duran más de los 5 minutos previstos en sus horarios. La columna más a la derecha representa la última hora estimada que se comunica a los usuarios mediante una notificación antes de que su cita sea ejecutada. Ésta servirá para comparar el tiempo de espera suponiendo que los usuarios siguen las indicaciones de cada notificación al pie de la letra, con el tiempo real de espera marcado en la Tabla 62, y que no es más que la resta de la primera columna a la segunda (ver Figura 60).

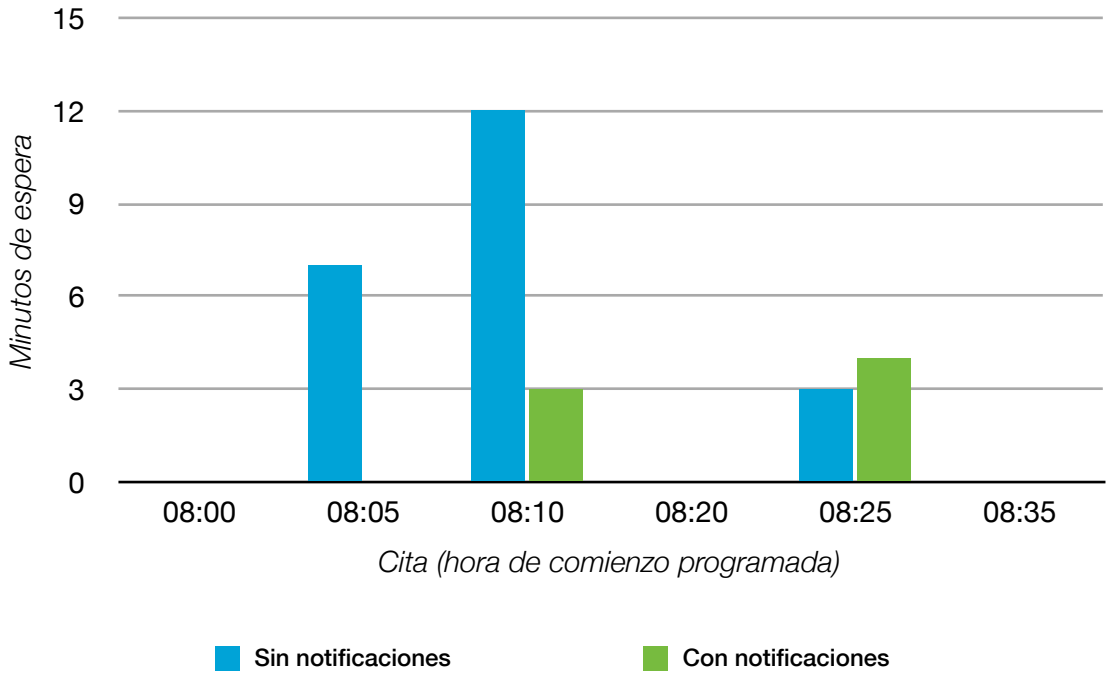

Figura 60. Comparación del tiempo de espera con y sin notificaciones de estado.

Notar que el tiempo de espera, en general, es menor con el control automático de colas que implementa este proyecto, que sin él. Si los usuarios se guían por las notificaciones que éste envía cada 5 minutos, no deberían de esperar más de esa

cantidad. Incluso si la frecuencia de envío de notificaciones fuese menor, se conseguiría una mayor precisión a cambio de ser alertado en más ocasiones.

Además, se ha comprobado como la ventana de tiempo establecida para la selección de las citas a notificar, ha variado su tamaño fijo de 30 minutos durante la simulación, en relación a la evolución en la cantidad del retraso. Si no hubiera sido así, las citas más alejadas en el momento de la ejecución del módulo de control no habrían sido notificadas.

Como segunda prueba se ha simulado una cola de citas que presente adelanto sobre el horario previsto. Para ello se ha a puesto a 3 el valor medio de probabilidad para los eventos: llegada de usuarios y duración de las citas. El resultado es el siguiente.

	Llegada del usuario	**Comienzo de la cita**	**Finalizado de la cita**
08:00 - 08:05	08:00	08:00	08:04
08:05 - 08:10	08:03	08:04	08:08
08:10 - 08:15	08:06	08:08	08:10
08:20 - 08:25	08:09	08:10	08:12
08:25 - 08:30	08:15	08:15	08:21
08:35 - 08:40	08:19	08:21	08:26

Tabla 63. Cola de citas generada aleatoriamente con adelanto.

En este caso, los resultados son difícilmente comparables como anteriormente. El módulo de control siempre tenderá a que las citas se ejecutan a su hora programada. Por ejemplo, la cita de las 08:05 recibirá una notificación a las 08:00 estimando que comenzará a su hora. Si el usuario sigue esa estimación debería de llegar a las 08:05. Por el contrario, el simulador establece que en el momento que acabe una cita y haya un usuario esperando en la cola (08:04), comienza automáticamente la cita reservada por éste, lo cual, reduce drásticamente el tiempo de espera en favor del no uso de notificaciones. Esto no significa que el módulo de control sea defectuoso, sino que impone una regla más estricta en el caso de haber adelanto en una cola: las citas comenzarán a su hora.

CAPÍTULO 7: CONCLUSIONES

En este capítulo se expondrán tanto las conclusiones generales del proyecto como las personales. Además, se hablará de las líneas futuras pensadas por el autor y los tutores en relación al sistema aquí proyectado.

7.1. Conclusiones generales del proyecto

Tras haber pasado casi un año desde que comenzase el análisis, diseño e implementación del proyecto, las conclusiones de éste no pueden ser más que positivas y satisfactorias.

Se ha conseguido con éxito llevar a cabo el objetivo principal marcado al comienzo del proyecto: desarrollar un sistema automatizado para el control de las colas de citas, que avise del estado de éstas a los usuarios, evitando de esta manera largas esperas o interrupciones en el servicio.

El eje principal del proyecto siempre a girado en torno al diseño de un proceso automatizado, que eliminase cualquier acción humana durante el notificado a usuarios del estado de una cola de citas. Esto ha sido posible, gracias a la utilización de una gran variedad de medios tecnológicos punteros: software avanzado de desarrollo (Xcode y Sublime Text Editor), hardware de última generación (iPhone y iPad), o lenguajes de programación modernos y potentes (Objective-C y Ruby).

Es importante destacar el especial cuidado que se ha tenido al proporcionar los mandos necesarios para los correspondientes operarios (empleados y administradores) en forma de aplicaciones web de gestión. Como éstas serán utilizadas durante el día a día, se ha intentado en todo momento que su funcionalidad sea sencilla y concisa, sin sobrecargarlas con opciones innecesarias que a la larga entorpecerían el flujo de trabajo de los mismos. De la misma manera, se ha actuado en el diseño e implementación de la aplicación móvil de servicio, con la que los usuarios finales podrán realizar diversas acciones relacionadas con la reserva, comprobación y cancelación de sus citas de una forma rápida e intuitiva.

El envío y recepción de notificaciones con el estado de una cola de citas y la estimación de cuando está previsto que comience la cita, se ha planteado del modo menos intrusivo para el usuario. Su frecuencia dependerá del tiempo medio de cita programado para la actividad a la que pertenece la cola, y por lo general, éste será alertado con una antelación de 30 minutos. Sólo en el caso de que exista un retraso, dicho valor aumentará, evitando posibles casos en los que los usuarios más espaciados en el tiempo dentro de una cola, no reciban notificaciones.

Por último, se quiere hacer especial mención a las funcionalidades de impresión de recibos y de visualización de ruta hasta el lugar de una cita, implementadas dentro de la aplicación móvil para usuarios finales. Ellas suponen un valor añadido importante

como elemento diferenciador, lo que en un futuro podría resultar en un volumen de ventas mucho mayor de lo estimado.

7.2. Conclusiones personales

A nivel personal, la realización de este proyecto ha supuesto un punto de inflexión en mis estudios, pues me ha ayudado a aprender una metodología de trabajo necesaria para llevar a cabo un análisis, un diseño y una implementación de un sistema automatizado de control y gestión.

Desde un principio, se planteó como un reto de enorme dificultad, pero que gracias a la ayuda de mis tutores, familiares, compañeros y amigos, he sido capaz de superar. Ha supuesto un año de trabajo muy intenso, que me ha permitido aplicar los numerosos conocimientos adquiridos durante mi estancia como estudiante en la Universidad Carlos III de Madrid, pero también aprender nuevas tecnologías como Ruby on Rails, que de cara a mi futura carrera profesional, espero que me ayude a afrontar nuevos retos.

Destacar también la satisfacción que me ha producido el hecho de construir un producto tangible, cuya principal finalidad sea la de intentar mejorar la calidad de vida de las personas. Es por ello por lo que he depositado grandes esperanzas en que el proyecto salga adelante y en un futuro, quién sabe, se convierta en algo cotidiano para cualquiera.

No acabará este apartado sin hacer especial mención, al incansable trabajo y esfuerzo demostrado por los tutores de este proyecto durante todo el año que ha supuesto su desarrollo. Su dedicación ha sido la pieza clave para que todo tomase la forma adecuada, y se consiguiesen los resultados esperados.

7.3. Líneas futuras

A continuación se describirá un conjunto de funcionalidades a considerar en un futuro para la mejora del sistema aquí planteado, pero que por razones de tiempo o de aprendizaje, se decidió por dejarlas fuera del proyecto durante esta primera fase.

7.3.1. Expansión a otras plataformas móviles

Como ya se comentó en el Capítulo 2, existen otras plataformas móviles a parte de iOS. La más conocida de ellas es Android, la cual está sufriendo un enorme crecimiento durante los últimos años, por lo que el desarrollo de una aplicación nativa para ese sistema operativo en concreto, supondría llegar a más usuarios finales, y en definitiva, aumentar los beneficios.

7.3.2. Estimación dinámica en el cálculo del estado

Durante la ejecución del módulo de control encargado de determinar el estado de una cola de citas, esto es, la cantidad en tiempo del retraso o adelanto, se realiza una operación de estimación sobre la hora de finalización de una cita en curso, atendiendo al tiempo medio asignado por cita. A partir de ahí, el algoritmo, por regla general, calcula el valor del estado mediante la diferencia entre dicha estimación y la hora en que realmente debería de haber acabado la cita.

Durante la etapa de análisis, se plantea la posibilidad de que esa estimación no sea constante, sino que dependa del histórico de cada usuario en el sistema, es decir, para aquellos usuarios que en citas anteriores tardaron más de lo programado durante su ejecución, el módulo de control estimará un mayor tiempo de duración para las posteriores. De igual manera ocurrirá para el caso en que hayan tardado menos de lo esperado, tal que el sistema realizará una estimación a la baja. También podría tenerse en cuenta otros parámetros del usuario para la estadística a realizar sobre la duración de las citas, como por ejemplo: su edad, su profesión, si llega tarde habitualmente, etc…

Sería interesante que el sistema reorganizase el orden de las citas de forma dinámica, atendiendo a la distancia que se encuentre cada usuario del lugar de la cita. Con la utilización del GPS incorporado en la mayoría de los *smartphones,* las aplicaciones móviles enviarían periódicamente la posición geográfica de los usuarios al servidor, dando este último mayor prioridad a aquellos que se encuentren más cerca del servicio.

Se decidió posponer la implementación de todas estas funcionalidades para fases futuras del proyecto, principalmente por la complejidad y privacidad del método.

7.3.3. Sistema empotrado

De cara a la venta del sistema completo a un cliente, sería conveniente ofrecerle la posibilidad de instalar un sistema empotrado en el lugar físico de cada actividad, con el que los usuarios finales pudiesen tanto reservar y cancelar citas, como comprobar el estado de una cola. Sería un dispositivo con pantalla táctil, que sirviese de interfaz al usuario para acometer cualquiera de esas operaciones. Haciendo un pequeño símil, equivaldría al cajero que puede encontrarse en la mayoría de la salas de cine, donde se pueden retirar las entradas compradas previamente a través de una página web.

Figura 61. Ejemplo de cajero automático para la reserva y compra de entradas de cine[39].

También, se ha contemplado para una fase posterior del proyecto, el conexionado del servidor con un sistema de pantallas informativas, previamente instaladas en el lugar físico donde se desarrolla la actividad del cliente, de manera que los usuarios, a medida que vayan llegando, estén correctamente informados del estado de la colas de citas.

Figura 62. Ejemplo de una pantalla informativa en sistemas con gestión de colas de citas.[40]

[39] Imagen obtenida en [35].

[40] Imagen obtenida en [36].

ANEXO A

ANEXO A: REFERENCIA BIBLIOGRÁFICAS

Todas las referencias aquí presentadas se encuentran revisadas a fecha de 1 de Junio de 2012.

[1] Lenguaje Unificado de Modelado UML : http://es.wikipedia.org/wiki/UML

[2] Diagramas de casos de uso: http://es.wikipedia.org/wiki/Diagrama_de_casos_de_uso

[3] Casos de uso UML : http://lsi.ugr.es/~mvega/docis/casos%20de%20uso.pdf

[4] Aplicación Mapas del iPhone : http://www.apple.com/es/iphone/features/maps-compass.html

[5] Apple Human Interface Guidelines : http://developer.apple.com/library/ios/documentation/userexperience/conceptual/mobilehig/MobileHIG.pdf

[6] AppStore en Wikipedia: http://es.wikipedia.org/wiki/App_Store

[7] App Store de Apple: http://www.apple.com/la/iphone/features/app-store.html

[8] Prepare Application for Submission: http://developer.apple.com/appstore/resources/submission/

[9] App Store Approval Process: http://developer.apple.com/appstore/resources/approval/

[10] App Store Pricing Matrix: http://www.app-machine.com/hmcms_media/public/files/ApplePreis_Matrix.pdf

[11] iAd Producer Software: http://developer.apple.com/iad/iadproducer/

[12] iAd de Apple: http://developer.apple.com/iad/

[13] Descarga del iOS Software Development Kit : http://itunes.apple.com/us/app/xcode/id448457090?mt=12

[14] Descarga del Android Software Development Kit: http://developer.android.com/sdk/index.html

[15] Herramientas del iOS SDK : http://developer.apple.com/technologies/tools/whats-new.html

[16] Herramientas del Android SDK: http://developer.android.com/guide/developing/tools/index.html

[17] Página web de Apple: http://www.apple.es

[18] XML : http://es.wikipedia.org/wiki/Extensible_Markup_Language

[19] JSON : http://www.json.org/

[20] UITabBarController : http://developer.apple.com/library/ios/#DOCUMENTATION/UIKit/Reference/UITabBarController_Class/Reference/Reference.html

[21] Diseño y automatización industrial: http://www.epsevg.upc.edu/hcd/material/lecturas/interfaz.pdf

[22] Whenever (programador de tareas en Ruby on Rails): https://github.com/javan/whenever

[23] UISplitViewController: http://developer.apple.com/library/ios/#documentation/uikit/reference/UISplitViewController_class/Reference/Reference.html

[24] CRUD en Ruby on Rails: http://guides.rubyonrails.org/routing.html#crud-verbs-and-actions

[25] Mercado global de SmartPhones: http://www.bgr.com/2011/05/19/android-grabs-53-of-global-smartphone-market-share-ios-50-of-application-revenues/

[26] Activaciones diarias de dispositivos Android: http://www.android.es/its-over-900-000-activaciones-diarias-de-dispositivos-android.html

[27] App Genome Report: https://www.mylookout.com/appgenome

[28] Lenguajes del lado servidor o cliente: http://www.adelat.org/media/docum/nuke_publico/lenguajes_del_lado_servidor_o_cliente.html

[29] PHP: http://www.php.net/manual/es/index.php

[30] App Store Downloads: http://www.apple.com/es/pr/library/2012/03/05Apples-App-Store-Downloads-Top-25-Billion.html

[31] iTunes de Apple: http://www.apple.com/es/itunes/

[32] Contenido de descarga DLC: http://es.wikipedia.org/wiki/Contenido_de_descarga

[33] Apple iAd Developers: http://advertising.apple.com/developers/

[34] Colegio Oficial de Ingenieros Industriales de Madrid: http://www.coiim.es/

[35] Cajero Servicaixa: http://11870.com/pro/cine-proyecciones/media/2f90978c

[36] Paneles informativos para sistemas de gestión de espera: www.inteled.info/es/aplicaciones.html